NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

SENSITIVITY ANALYSIS OF THE TOPOLOGY OF CLASSIFICATION TREES

by

Izumi Kobayashi

December 1999

Thesis Advisor: Samuel E. Buttrey
Second Reader: Robert A. Koyak

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1999	Master's Thesis

4. TITLE AND SUBTITLE
SENSITIVITY ANALYSIS OF THE TOPOLOGY OF CLASSIFICATION TREES

5. FUNDING NUMBERS

6. AUTHOR(S)
Kobayashi, Izumi

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(maximum 200 words)***

The use of classification trees is one of the most widely used techniques in classification. It is well known that classification trees are not stable in their topology, in contrast to their robustness with respect to misclassification rate.

This thesis defines a measure that compares the topology of two trees and studies how a tree's topology changes when the dependent (Y) variable or the independent (X) variables are perturbed. This allows us to examine the "robustness" of tree topology under perturbation and to compare it to the robustness with respect to the misclassification rate under the same perturbations.

We show that the tree topology can change significantly even for small perturbations in many sets of data. This suggests that even small measurement errors in the variables can affect the tree topology greatly. Because data are often measured with error, it follows that splitting rules in trees may not be suitable for use in making policy decisions. We propose a measure for tree topology, and show that tree topology changes faster than the misclassification rate does under mild perturbations. This finding formalizes the concept that tree models are more stable in terms of misclassification rate than in terms of topology.

14. SUBJECT TERMS
Classification Tree, Sensitivity Analysis

15. NUMBER OF PAGES
78

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFI- CATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

SENSITIVITY ANALYSIS OF THE TOPOLOGY OF CLASSIFICATION TREES

Izumi Kobayashi
Technical Official 2nd Grade, Japan Defense Agency
M.S., Ochanomizu University, 1992

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
December 1999

Author: Izumi Kobayashi
Izumi Kobayashi

Approved by: Samuel E. Buttrey
Samuel E. Buttrey, Thesis Advisor

Robert A. Koyak
Robert A. Koyak, Second Reader

Richard E. Rosenthal
Richard E. Rosenthal, Chairman
Department of Operations Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The use of classification trees is one of the most widely used techniques in classification. It is well known that classification trees are not stable in their topology, in contrast to their robustness with respect to misclassification rate.

This thesis defines a measure that compares the topology of two trees and studies how a tree's topology changes when the dependent (Y) variable or the independent (X) variables are perturbed. This allows us to examine the "robustness" of tree topology under perturbation and to compare it to the robustness with respect to the misclassification rate under the same perturbations.

We show that the tree topology can change significantly even for small perturbations in many sets of data. This suggests that even small measurement errors in the variables can affect the tree topology greatly. Because data are often measured with error, it follows that splitting rules in trees may not be suitable for use in making policy decisions. We propose a measure for tree topology, and show that tree topology changes faster than the misclassification rate does under mild perturbations. This finding formalizes the concept that tree models are more stable in terms of misclassification rate than in terms of topology.

THESIS DISCLAIMER

The reader is cautioned that the computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

I. INTRODUCTION 1

A. CLASSIFICATION TREES 1
1. General 1
2. Robustness of Tree Models 4
B. CONSTRUCTION OF CLASSIFICATION TREE MODELS 5
1. Growing Trees 5
2. Getting the Right Size 6
3. Construction of Trees in This Thesis 8
C. EXISTING TECHNIQUES FOR SENSITIVITY ANALYSIS 9
D. THE FOCUS OF THE THESIS 10

II. DEFINITION OF "TREE TOPOLOGY ROBUSTNESS" 11

A. DEFINITION OF THE MEASURE OF "TREE TOPOLOGY ROBUSTNESS" ... 11
1. Definition 11
2. Examples 12
3. Notes on the Measure of Topology Difference 15
B. DEFINITION OF THE MEASURE OF "TREE TOPOLOGY ROBUSTNESS" WHEN PERTURBING CONTINUOUS *X* 16
1. Definition 17
2. Examples 18
3. Note 20

III. SIMULATION AND RESULTS 21

A. DATA 21
1. Selection Criteria 21
2. Summary 22
3. Detailed Descriptions 22
B. PERTURBATION TECHNIQUES 25
1. Categorical Data 25
2. Continuous Data 25
C. SIMULATION 26
1. General 26
2. Programming Language 26
3. Treatment in the Case of Continuous X Variables 26
D. CONSTRUCTION OF ORIGINAL TREES 26
1. Selection Rule 26
2. Original Trees for Each Data Set 27
E. SIMULATION RESULTS AND DISCUSSION 28
1. Perturbation of Y 28
2. Perturbation of X 32

IV. CONCLUSIONS **35**
A. FURTHER RESEARCH 36
APPENDIX A. CHANGES IN STRUCTURE **37**
A. PERTURBATION OF *Y* 37
1. Number of Terminal Nodes *37*
2. Depth *40*
B. PERTURBATION OF *X* 42
1. Number of Terminal Nodes *42*
2. Depth *44*
APPENDIX B. S-PLUS CODE **47**
A. FUNCTION FOR PERTURBING *Y* 47
B. FUNCTION FOR PERTURBING *X* 53
C. FUNCTION FOR PERTURBING CATEGORICAL VALUE 62
LIST OF REFERENCES **63**
INITIAL DISTRIBUTION LIST **65**

EXCECUTIVE SUMMARY

The use of classification trees is one of the most widely used techniques in classification. It is well known that classification trees are not stable in their topology, under mild perturbations of the data, in contrast to their robustness with respect to misclassification rate.

Since the focus in these models has been on misclassification rate, little work has been done on examining their robustness with respect to topology. This thesis defines a measure that compares the topology of two trees and studies how a tree's topology changes when the dependent (Y) variable or independent (X) variables are perturbed. This allows us to examine the "robustness" of tree topology under perturbation and to compare it to the robustness with respect to the misclassification rate under the same perturbations.

A measure of "change in tree topology" is defined based on the depth and number of differences in splitting rules. When one tree differs from another at a particular node A, the difference in topology is defined as the ratio of the number of potential nodes under and including A to the total number of potential nodes in the tree. Differences at nodes that are descendants of A are ignored. Then the total difference in tree topology is the sum of all individual differences. This yields a measure that reaches its maximum value of 1 when the two trees have different splits at the root node and takes smaller values as differences are observed farther down the tree.

Several sets of data are used to simulate how the tree topology changes when either the Y or the X variables are perturbed. The results show that the tree topology changes significantly even for small perturbation rates in most sets of data. This suggests that even small measurement errors could affect the tree topology greatly. Because data are often measured with error, it follows that splitting rules in trees are not suitable to be used to make policy decisions. That is, basing a decision on whether a particular variable appears at the root of the tree is a poor idea, since often that variable might change if the data are perturbed only slightly. The simulation results also show that the total difference in tree topology increases at a much higher speed than the misclassification rate does, which implies that tree models are more stable in terms of the misclassification rate than in topology.

ACKNOWLEDGMENT

The author wants to thank Professor Buttrey for his guidance and patience during the work. The author is also grateful to Professor Koyak for his valuable advice.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. CLASSIFICATION TREES

1. General

In a classification problem we are given a *learning set* of data $(Y_i, \mathbf{X}_i)$, $i = 1, \ldots, n$, in which the Y_i are class labels. The vector $\mathbf{X}_i$ may have numeric or categorical values. The object is to build a rule by which to classify new observations from a so-called *test set*. One of the most widely used techniques in classification is the classification tree. Classification trees were originally developed by social scientists and were used in regression in the early 1960s. The work on trees by Breiman and Friedman began in 1973 and their work was published in 1984.

A classification tree is a simple model for classification. It is constructed by recursive splitting of a subset of the measurement space S into two descendant subsets, beginning with S itself. A "split" consists of a division, formed by conditions on one of the $\mathbf{X}$ variables, of the data into two parts. For example, Figure 1 shows a classification tree built with the well-known "Iris" data (see Section III.A.3). In this example the $\mathbf{X}$ variables are measurements on certain iris flowers, and the Y is a label corresponding to one of the three species represented in the data. A split might be, for instance, the following: (petal length < 5 mm, petal length ≥ 5 mm). Assuming no missing data, every item in S clearly falls into exactly one of these two groups.

By convention, the root node of a tree is called node 1 and the left and the right child nodes of node j are called nodes $2j$ and $(2j + 1)$ respectively. The "depth" of a node is one more than the number of splits traversed to travel down the tree from the root to that node. By definition, the root node is at depth 1. The child nodes of a node at depth d are at depth $(d + 1)$. The depth of node j is computed as $d = [\log_2(j)] + 1$, where $[x]$ = the greatest integer that does not exceed x. The "total depth" of a tree is the maximum depth of the tree.

The terminal nodes – those that are not split – are known as "leaves." Leaves are disjoint from one another and define a partition of S. That is, every element of the learning set falls into exactly one leaf. Each leaf has attached to it a class label, determined by plurality voting among the cases of the learning set falling into that node. For example, if a leaf has fifteen observations of the learning set and seven of them have class label A, five have B and three have C, then that leaf is associated with class label A. Each non-terminal node contains a splitting rule consisting of the variable and the splitting value. Those cases in node j answering "yes" go to the left child node and those answering "no" go to the right child node.

In the example in Figure 1, each terminal node is indicated by a rectangular box, while each non-terminal node is indicated by an ellipse. The number under each node shows the number of that node. As mentioned above, each terminal node has a class label attached to it. For example, node 2 has the label *Setosa*. Each non-terminal node has a splitting rule. For instance, node 6 contains the splitting rule "*Petal.L.* < *4.95*." Every such rule consists of a variable (e.g. *Petal. L.*) and the splitting value (e.g. "< 4.95").

We are now in a position to use the classification tree on new data. For instance, if a new case had *Petal. L.* = 2.6 and *Petal. W.* = 1.2, it would go right at the root node, left at node 3, left at node 6 and fall into node 12. Since the label of node 12 is *Versicolor*, this new case would be classified as *Versicolor*.

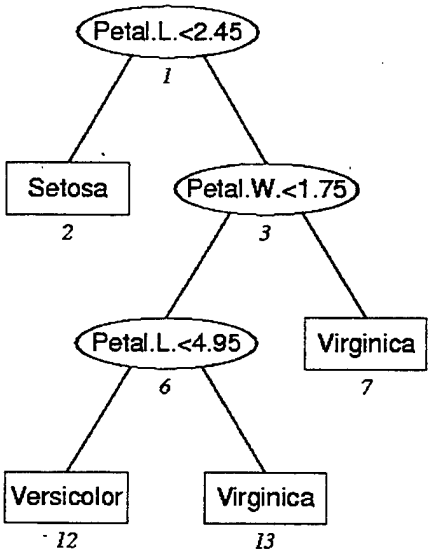


Figure 1: Example of Classification Tree. Those cases in node *j* answering "yes" go to the left child node and those answering "no" go to the right node. For example, if a new case had *Petal. L.* = 2.6, it would go right at the root node. This classification tree was constructed with the well-known "Iris" data (see Section III. A. 3.)

Breiman, Freidman, Olsen and Stone (1984) pointed out the advantages of a tree-structured approach:

- It can be applied equally well to categorical or continuous data.
- It makes powerful use of conditional information in handling nonhomogeneous relationships.
- It does automatic stepwise variable selection and complexity reduction.

- It is invariant under all monotone transformations of individual ordered variables.
- The tree procedure output gives easily understood and interpreted information regarding the predictive structure of the data.

2. Robustness of Tree Models

It is well known that the classification tree is not stable in its topology. Breiman (1984) introduced two examples that showed this instability. In each example, four sets of data were randomly generated on a computer in exactly the same way but using different random number seeds. Then, classification trees were built with each set of data. The example demonstrated that two trees constructed with different sets of data could have different variables splitting at the root node even though the data were generated with exactly the same rule. The authors suggested that "tree instability is not nearly as pronounced" in practice as in their examples. They also suggested that "adding small amounts of noise to the data does not appreciably change the tree structure, except, perhaps, for a few of the lower nodes."

In contrast to the instability of its topology, the classification tree is fairly stable in the misclassification rate. Breiman (1984) showed that the misclassification rates were not very different for all the data sets used in the examples described above. Although the book by Breiman *et al* was published in the early 1980s, robustness in topology has not been studied much as most interest has concentrated on the misclassification rate.

B. CONSTRUCTION OF CLASSIFICATION TREE MODELS

The steps to constructing a tree model are the following: (i) growing a tree, and (ii) pruning it to the right size. The details of each step are described below.

1. Growing Trees

A tree is grown by splitting each node into two descendant nodes using the splitting rule that most reduces the "impurity" of the node, starting with the root node. First a measure of the impurity of a node needs to be defined. A node is "pure" if all the observations from the learning set that fall into that node have the same Y value. Any impurity measure should give such a node an impurity score of zero. This thesis used the S-Plus (MathSoft, 1998) software in which the impurity measure is the deviance, defined below. While other measures of impurity, such as the Gini index and the entropy, have been proposed, "the overall misclassification rate of the tree constructed is not sensitive to the choice of a splitting rule" according to Breiman (1984).

The tree is grown by splitting each node using the splitting rule that most reduces the deviance. Deviance D is defined by viewing a tree as a probability model where at each terminal node j, the data constitute a random sample from the multinomial distribution with probabilities p_{jk} over the k classes (Y labels). Assuming that we have a random sample of size n_{jk} from the multinomial distribution specified by p_{jk}, the likelihood of the data is proportional to

$$\prod_{\substack{\text{terminal} \\ \text{nodes } j}} \prod_{\text{classes } k} p_{jk}^{\,n_{jk}}.$$

Using this likelihood, the deviance is defined as –2 times the sum of the log-likelihoods:

$$D = \sum_{\substack{\text{terminal} \\ \text{nodes } j}} D_j, \quad \text{where } D_j = -2 \sum_{\text{classes } k} n_{jk} \log p_{jk} .$$

Because the p_{jk}'s are not known, they are estimated by the proportions of the k classes in node j.

At each node s, splitting it into nodes $2s$ and $2s+1$ changes the probability model within node s. Thus, the reduction in deviance for the tree is given by $D_s - D_{2s} - D_{2s+1}$. The split which reduces the deviance the most among all possible splits at node s is chosen as the next split.

The growing step is continued until each terminal node is "pure enough" or the number of cases falling into each terminal node is "small."

2. Getting the Right Size

A grown tree is usually very large and too closely fit to the data. For example, if a tree is grown until each terminal node contains only one case, the resulting tree will have a resubstitution error estimate of zero, but the tree will have little predictive power for a new data set from the same sampling distribution. Here, the resubstitution error estimate is defined by the proportion of the learning set cases that are misclassified by the tree built with the learning data set. As suggested in Breiman (1984) "too large a tree will have a higher true misclassification rate than the right-sized tree." On the other hand, too small a tree also will have a higher true misclassification rate than the right-sized tree since it will not use some of the information available in the training data set.

Therefore, the objective is to find the "right-sized" tree. Breiman (1984) proposed combination of minimal cost-complexity pruning and cross-validation. In minimal cost-complexity pruning, the cost complexity of a sub-tree T_s (that is, a smaller tree with the same root node) of a tree T for some parameter α is defined as $C_\alpha(T_s) = R(T_s) + \alpha l(T_s)$, where $R(T_s)$ is the resubstitution error of T_s defined by the proportion of learning set items that are misclassified by the sub-tree and $l(T_s)$ is the size of T_s as measured by the number of terminal nodes. Thus $C_\alpha(T_s)$ is a linear combination of the sub-tree's error estimate and its size. It can be shown that there is a unique sub-tree T_s that minimizes cost complexity for any value of α. That is, for any sub-tree $T_{s'}$ that has T_s as a pruned sub-tree, either $C_\alpha(T_{s'}) > C_\alpha(T_s)$ or $C_\alpha(T_{s'}) = C_\alpha(T_s)$ must hold. Using this fact, an increasing sequence $\{\alpha_k\}$ with $\alpha_1 = 0$ can be found that has the property that, for each α_k, T_k is the tree which minimize the cost complexity for α_k and T_k is a sub-tree of T_{k-1}, where $T_1 = T$. Since the number of terminal nodes of the grown tree T is finite, the sequence $\{\alpha_k\}$ is a finite sequence. The sequence $\{\alpha_k\}$ is used in the next step.

The next step is selecting a single tree based on its misclassification rate. Since the resubstitution estimate is known to underestimate the true misclassification rate, the test sample is used to obtain a less biased estimate if the test data set is large enough to be divided into a learning sample and a test sample. For smaller data sets that have about 200 to 1000 observations, cross-validation can be used. In cross-validation, the original data are randomly split into n equal subsamples denoted by S_1, S_2, ..., S_n. A typical choice is $n = 10$. For each i, $S-S_i$ (the set of observations excluding those in subsample

S_i) is used as a training set to build a tree, with S_i used as the test set to calculate the error estimate R_i. The cross-validation error estimate $R_\alpha^{CV}(T)$ is computed as the mean of the R_i for each α. Finally, the tree size corresponding to α which minimizes $R_\alpha^{CV}(T)$ is chosen as the optimal tree size.

Unlike the original definition of Breiman (1984), the default setting of S-Plus uses the deviance defined in the previous section instead of the misclassification rate to calculate $R(T)$. This thesis similarly uses the deviance to determine the optimal size. Once the optimal size of the grown tree is estimated, the sub-tree of that size is selected as the final tree model.

3. Construction of Trees in This Thesis

For this thesis, classification trees were constructed from data as follows. First, a tree was grown so that the terminal nodes of the tree were as pure as possible (subject to S-Plus' default stopping rules) using the S-Plus function `tree()`. Next, cross-validation was used to estimate the optimal size. The size for which the tree had the smallest cross-validated deviance was chosen as the optimal size. In the case of a tie in the deviance, the smallest size was chosen as optimal. The function `cv.tree()` was used for cross-validation. Finally, the tree was pruned to that size using `prune.tree()`.

Note that when the child nodes t_1, t_2 from the same parent node p have the same class labels, the misclassification rate is the same as if p were a terminal node, but the deviance is different. If we were only interested in the misclassification rate, we could

always prune off these child nodes t_1 and t_2 with no increase in the misclassification rate. However, in this thesis, we do not do this.

C. EXISTING TECHNIQUES FOR SENSITIVITY ANALYSIS

There have not been many studies conducted on measurement error in categorical data analysis. On the other hand, much work has been done on sensitivity analysis in linear regression. Chatterjee and Hadi (1988) introduce three different approaches to assess the effects of measurement errors in the explanatory variables: the asymptotic approach, the perturbation approach and the simulation approach. The first two approaches are analytical while the last one is empirical.

In the asymptotic approach, the effects of errors on the regression coefficients are examined as the sample size is increased indefinitely. In the perturbation approach, an upper bound for the relative errors in the estimated regression coefficients can often be obtained analytically when the data are perturbed by small amounts. The latter approach, the effect of measurement errors in only one variable can be studied.

In contrast, the joint effects of measurement errors occurring in multiple variables can be studied in the simulation approach. In this approach, the measurement errors are generated randomly on a computer, and the data are perturbed by these errors. The behavior of the regression coefficients is then studied under these perturbations.

In this thesis, the simulation approach was used. That is, tree models were built from empirically perturbed data and the differences in topology from the original tree

model were measured. The perturbation methods that were used will be described in section III. B.

D. THE FOCUS OF THE THESIS

Although it is well known that tree topology can be unstable, robustness in topology has not been studied much because most interest has concentrated on the misclassification rate. This thesis defines a measure that compares the topology of two trees and studies how a tree's topology changes when either the dependent (Y) variable or independent (X) variables are perturbed. This thesis also compares robustness with respect to the tree topology to robustness with respect to the misclassification rate under the same perturbations.

II. DEFINITION OF "TREE TOPOLOGY ROBUSTNESS"

A measure of "tree topology robustness" is defined in this chapter based on the concept of tree depth and differences in splitting rules. A general definition and examples are given in section A. In section B, a special case is considered in which continuous **X** variables are perturbed. The latter case requires more careful treatment since the splitting value in the perturbed tree will almost always be different from the original one at the root node, even if all cases fall into exactly the same nodes as in the original tree. As a result, if the measure defined in section A is used, the total difference in topology will always be 1, although the topologies of the two trees may be similar.

A. DEFINITION OF THE MEASURE OF "TREE TOPOLOGY ROBUSTNESS"

1. Definition

Let A be the original tree and let B be a tree compared to A. Suppose that the total depth of tree A is d_A. For node j of A, $d_A(j)$ denotes the depth of j and $A(j)$ denotes the sub-tree that has j as its root node. Note that $A(1) = A$. The depth $d_A(j)$ of j is calculated by $d_A(j) = [log_2(j)] + 1$, where $[x]$ = the greatest integer that does not exceed x.

When a difference in the splitting rule, consisting of either a difference in the splitting variable or in the splitting value for the same variable, is observed between two trees at node j, the difference in topology Δ_j is defined by

$$\Delta_j = \frac{2^{d_A - d_A(j)+1} - 1}{2^{d_A} - 1} = \frac{\textit{the total number of potential nodes of tree } A(j)}{\textit{the total number of potential nodes of tree } A}.$$

The difference in topology Δ_j has the following properties:

- If the difference is observed at the root node, then $\Delta_1 = 1$;
- If node j and node k of tree A are at the same depth, then $\Delta_j = \Delta_k$;
- If node k is one level below that of node j (i.e., $d_A(k) = d_A(j) + 1$), $\Delta_k \approx \Delta_j /2$.

The total difference in topology D between two trees is defined as follows: Let J be the set of the nodes at which the splitting rules differ between two trees. If node j belongs to J, then none of j's descendants belong to J. If no difference is observed between two trees, J is empty. The total difference between two trees is defined by

$$D(A,B)=\sum_{j\in J}\Delta_j=\sum_{j\in J}\frac{2^{d_A-d_A(j)+1}-1}{2^{d_A}-1}.$$

2. Examples

Simple examples of the measure defined in the previous section are given here. Each example is based on the Glass data which has 214 observations consisting of seven classes, and in which all nine independent variables are continuous. A detailed description of this data is given in Section III. A. 3. As in the examples presented in Chapter I, rectangular boxes indicate terminal nodes while ellipses indicate non-terminal nodes. The number in each terminal node indicates the class of a case falling into that node and the number under a node indicates the node number.

Trees A and B in Figure 2 were constructed by perturbing the Y variable with perturbation rate 0.002 using different random number seeds. The perturbation technique will be described in Section III. B. In tree A, node 2 has the splitting rule $Na < 13.785$ while node 2 has the splitting rule $Ba < 0.335$ in tree B. Node 2 is at depth 2 and the total depth of tree A is 3. Therefore, the total difference in topology is $D(A, B) = \Delta_2 = (2^{3-2+1} - 1) / (2^3 - 1) = 3/7$.

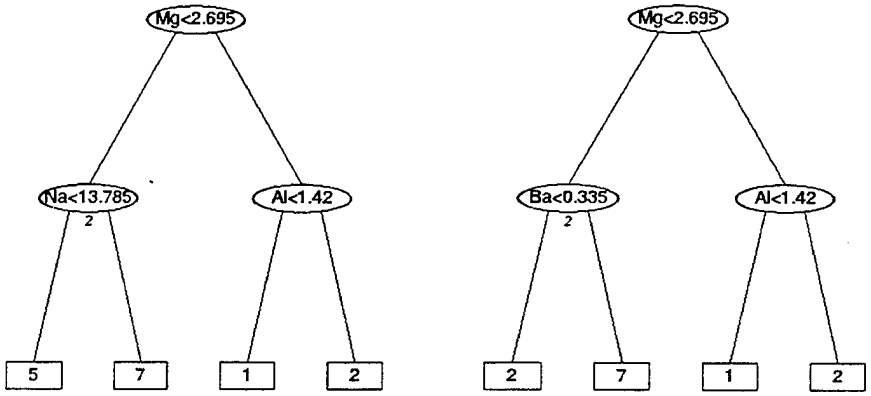


Figure 2: Tree A (left) and Tree B (right). The splitting rules differ at node 2.

In Figure 3, tree C was built using the original Glass data and tree D was built using the Glass data in which the Y variable was perturbed with perturbation rate 0.002. Although node 6 is a terminal node in tree C, in tree D it appears as a non-terminal node with the splitting rule $RI < 1.51705$. Since the total depth of tree C is 4 and node 6 is at depth 3, the total difference in topology between trees C and D is calculated as $D(C, D) = \Delta_6 = (2^{4-3+1} - 1) / (2^4 - 1) = 3/15$.

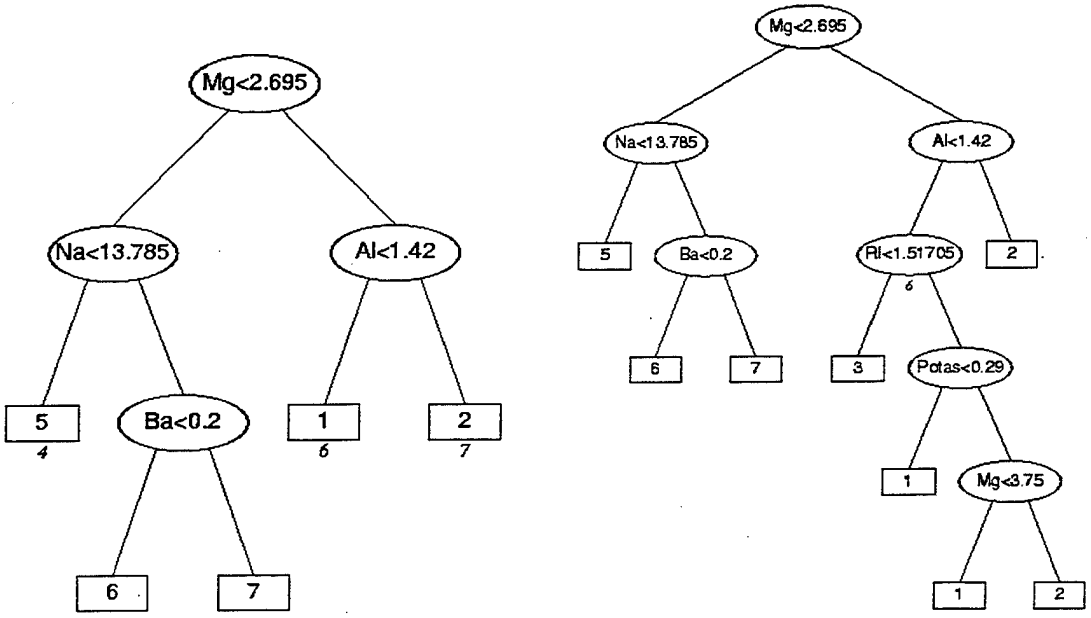


Figure 3: Tree *C* (left) and Tree *D* (right). Node 6 is a terminal node in tree *C* while node 6 is a non-terminal node in tree *D*.

Tree E in Figure 4 was built using the Glass data in which the Y variable was perturbed with perturbation rate 0.002. When compared to tree C, differences can be observed at nodes 4, 6 and 7. Since these nodes are at depth 3 and the total depth of tree C is 4, the total difference in topology between tree C and E is $D(C, E) = \Delta_4 + \Delta_6 + \Delta_7 = 3(2^{4-3+1} - 1) / (2^4 - 1) = (3)(3)/15 = 9/15$.

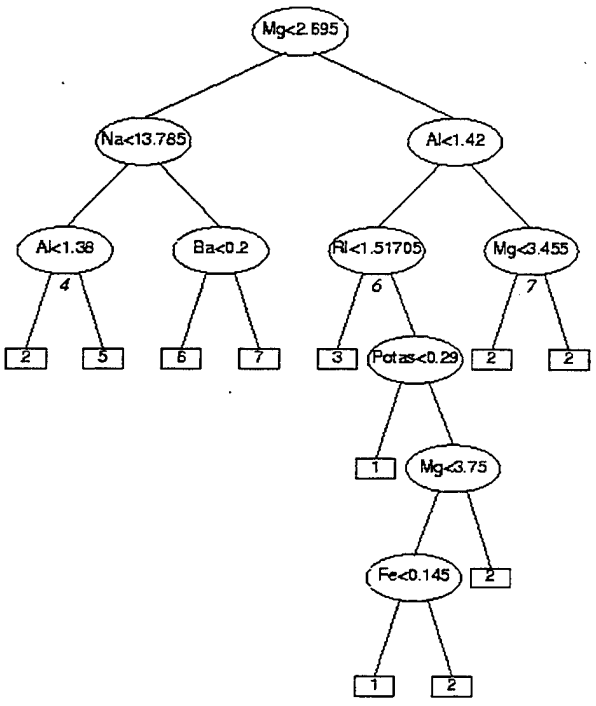


Figure 4: Tree *E*. Nodes 4, 6 and 7 are non-terminal nodes in tree *E* while these nodes are terminal nodes in tree *C*.

3. Notes on the Measure of Topology Difference

Given two trees T_1 and T_2, $D(T_1, T_2)$ and $D(T_2, T_1)$ will not, generally, be equal. In the above examples, $D(B, A) = \Delta_2 = (2^{3-2+1} - 1) / (2^3 - 1) = 3/7 = D(A, B)$; however, since the total depth of tree D is 6, $D(D, C) = \Delta_6 = (2^{6-3+1} - 1) / (2^6 - 1) = 15/31 \neq 3/15 =$

$D(C, D)$. We do not consider this overly worrisome because the measure is at its heart asymmetric; there is always an "original" tree and a tree built from perturbed data.

When the difference in topology between two trees is considered, there are two possible kinds of differences: the difference in the splitting rule (at a certain node) and the difference in the structure. However, the difference in the structure between two trees is not considered in this thesis. For instance, in the previous section, we had $D(C, D) = 3/15$ when comparing tree C and D. If we compare tree A to tree C, we also have $D(C, A) = \Delta_5 = (2^{4-3+1} - 1) / (2^4 - 1) = 3/15 = D(C, D)$, even though it appears to the eye that the difference between trees C and D is larger than that between trees C and A.

In addition, the case where the splitting variable is different, and the case where the splitting variable is the same but the value is different, both produce the same measure of topology difference although a difference in variable is considered to have the stronger effect on the difference in tree topology. Perhaps an alternative measure of topology difference can be designed to address this concern.

B. DEFINITION OF THE MEASURE OF "TREE TOPOLOGY ROBUSTNESS" WHEN PERTURBING CONTINUOUS X

In this section, the definition given in section A is modified for the case where the X variables are continuous and subject to perturbation. As described above, the splitting value in the perturbed tree will almost always be different from the original one at the root node; thus the total difference in topology under the earlier measure will often be equal to 1 even when the topologies of the two trees are similar. To avoid this problem,

the difference in topology when continuous X is perturbed is defined in the manner described below.

1. Definition

When two trees differ at node j in the splitting variable, the difference in topology Δ_j is defined in the same way as the case of categorical X variables. If the two nodes have the same splitting variable at node j but the splitting values are different at node j, the difference in the proportions of cases in the two trees falling into the child nodes of node j is used to adjust the difference in the topology. In this adjustment, the difference in topology is defined so that the more different the number of each class is at the child nodes of node j between the two trees, the greater the difference in topology.

Let n_{jk} be the number of cases of class k falling into the node j, where $k = 1, 2, \ldots, K$. The total number of cases falling into node j is $n_j = \sum_{k=1}^{K} n_{jk}$. Then, the difference in the components is calculated by $c_j = \sum_{k=1}^{K} \left| n_{jk}{}^{A} - n_{jk}{}^{B} \right|$, where $n_{jk}{}^{T}$ = the number of cases of class k in the node j of the tree T. Using c_j, Δ_j is defined by,

$$\Delta_j = \begin{cases} \dfrac{2^{d_A - d_A(j)+1} - 1 - \left(1 - c_j / n_j\right)}{2^{d_A} - 1} & \text{if } c_j \neq 0 \\ 0 & \text{if } c_j = 0 \end{cases},$$

where c_j is the total number of cases switching from the node $2j$ (the left child node) to the node $(2j + 1)$ (right child node) and from the node $(2j + 1)$ to the node $2j$. Note that this measure does not account for the case of switching nodes within the same class. That

is, if n cases of class k move from the node $2j$ to the node $(2j + 1)$ but the same number of cases of class k move from node $(2j + 1)$ to node $2j$, then $c_j = 0$.

Next, the set J of nodes at which the topology changes is constructed in the following way. If the splitting variable changes at node j, then j is a member of J. If the variable is the same but the splitting value is different, $j \in J$ if $c_j \neq 0$, and $j \notin J$ if $c_j = 0$. Furthermore, as in the case of the categorical X variables, if the node $j \in J$, the descendant nodes of node j do not belong to J. Once the set J is found, the total difference in the topology is assigned using $D(A,B)=\sum_{j\in J}\Delta_j$.

2. Examples

The following figures give examples of how the total difference in topology $D(A, B)$ between two trees A and B is calculated. Tree F was constructed using the original Wine data set, while trees G and H were built from the data sets obtained by perturbing the X variable with the perturbation rate 0.01 with different random number seeds. The Wine data set consists of 178 cases from three classes of wines, and each case has thirteen continuous attributes. The perturbation technique for the continuous variables is described in Section III. B. 2. Numbers under nodes indicate the number of cases of each class falling into that node. For example, (59 71 48) under the root node means that 59 cases of class 1, 71 cases of class 2 and 48 cases of class 3 are in that node.

Trees F and G have the same splitting variable but different splitting values at the root node. Checking their child nodes (2 and 3), we observe that the number of class-2 cases are different between two trees: there are 14 class-2 cases at node 2 of tree F, but 15

cases of the same class in tree G. That is, one case of class 2 switches from node 3 to node 2. Thus, the total difference in topology is $D(F, G) = \Delta_1 = (2^{4-1+1} - 1 - (1 - 1/178))/(2^4 - 1) = 0.9337$.

Tree H also has a splitting value different from tree F at the root node. In this case, however, since the number of cases falling into the child nodes is the same between two trees, the difference in topology at the root node is $\Delta_1 = 0$. Similarly, $\Delta_2 = \Delta_3 = 0$. Going farther down the nodes, differences are observed at node 5, 6 and 7: they are non-terminal nodes in tree F but terminal nodes in tree H. Therefore, the total difference is $D(F, H) = \Delta_5 + \Delta_6 + \Delta_7 = 3(2^{4-3+1} - 1)/(2^4 - 1) = 3(3)/15 = 0.6$.

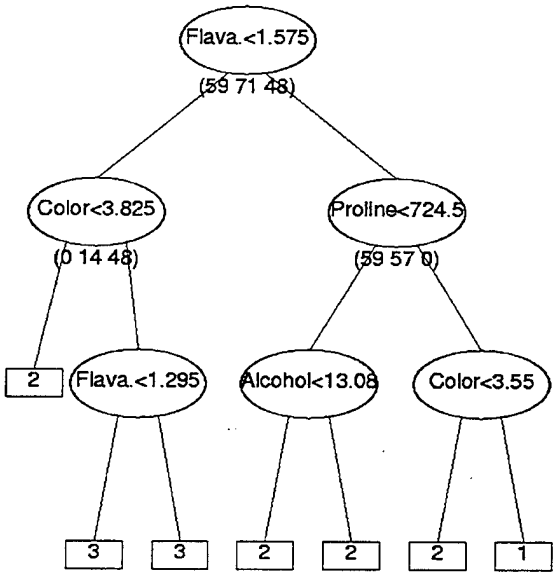


Figure 5: Tree *F*. This tree was built with the original Wine data set. Numbers under nodes show the number of cases of each class falling into that node.

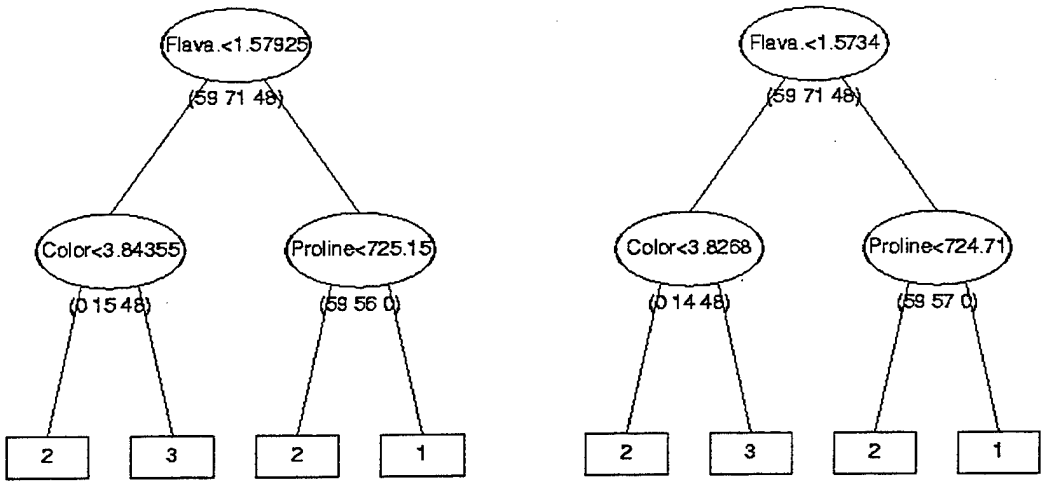


Figure 6: Trees *G* (left) and *H* (right). These trees were constructed using the Wine data in which the *X* variables were perturbed slightly with different random number seeds. Both trees have splitting values at the root nodes different from tree *F*. However, the number of the cases falling into nodes 2 and 3 are different between trees *F* and *G* while they are the same between trees *F* and *H*.

3. Note

The above examples also point out a problem of the measurement. In the first example, the total difference in topology is quite high, although trees *F* and *G* look alike at the first three nodes. Furthermore, in spite of the similarity in topology between trees *G* and *H*, the differences from tree *F* of each tree, $D(F, G)$ and $D(F, H)$, are not the same. These facts suggest that the total difference in topology could be very high even though two trees look alike. This is because changes at the root node always result in large differences in topology, even if those changes may not be apparent.

III. SIMULATION AND RESULTS

This section describes the data we used in our simulation, the techniques we used to perturb the data, the simulations themselves, and the results. In our simulations, we constructed a best-fitting tree from some well-known data sets; then we perturbed the data (either the Y variable or the X variables) slightly and built a new tree using the perturbed data. Our criterion of interest is the change in tree topology as a function of the size of the perturbation. Appendix B presents the S-Plus code that was used for computations.

A. DATA

1. Selection Criteria

For simplicity, data sets with the following properties were used for simulation:

- "Middle-sized" data sets, having between 150 to 1000 observations;
- Type of independent (X) variables: all independent variables are the same type (either all categorical or all continuous) for each set of data;
- No missing data.

2. Summary

A summary of the data sets used is given in the following table:

Data name	Classes	Cases	No. of X variables	Type of X
Digits1*	10	500	7	Categorical
Digits2*	10	500	7	Categorical
DNA	3	1000	60	Categorical
Diabetes	2	768	8	Continuous
Glass	7	214	9	Continuous
Iris	3	150	4	Continuous
Sonar	2	208	60	Continuous
Wine	3	178	13	Continuous

(*) Digits1 is used for the perturbation of Y variable and Digits2 is used for the perturbation of X variables.

Table 1: Summary of the Data Set (see text for references)

3. Detailed Descriptions

The data sets used in this thesis are described below:

a) Digits

These simulated data, based on the seven segments that compose an electronic image of a digit, were introduced to test classification trees in Breiman (1984). The 500 cases were generated using the same technique as the original authors. The data set contains ten classes (the ten digits) and seven binary attributes (the segments that are either on or off). Breiman (1984) constructed the data so that each of the X values is in error, independently of the others, with probability 0.1. When investigating the effects of perturbing Y, our data followed the same rule. When investigating the effects of perturbing X, each X variable originally had no error.

b) DNA

These data were downloaded from the UC Irvine Data Repository. The original data had 3,190 cases with three classes and 60 categorical attributes. Each case is a set of 60 bases from a DNA sequence, and the class describes a type of "junction" (named "EI," "IE," or "neither") in the middle of the set of 60. Because of memory limits in the computers running S-Plus, 1,000 cases were chosen randomly and used in this thesis. Originally, classes were distributed as 767 of EI, 768 of IE and 1655 of neither. In the set used in this thesis, classes were distributed as 222 of EI, 226 of IE and 552 of neither.

c) Diabetes

These data were also downloaded from the UC Irvine Data Repository. The data had 768 cases representing Pima (Native American) females, of which 500 cases were class 0 (not having diabetes) and 268 cases were class 1 (having diabetes). The eight attributes are continuous and are as follows: 1) Number of times pregnant, 2) Plasma glucose concentration a two hours in an oral glucose tolerance test, 3) Diastolic blood pressure, 4) Triceps skin fold thickness, 5) 2-Hour serum insulin, 6) Body mass index (weight in kg/(height in m)^2), 7) Diabetes pedigree function and 8) Age.

d) Iris

This data set is built into S-Plus. The data consist of measurements on 150 flowers, 50 from each of 3 iris species *Setosa*, *Versicolor*, and *Virginica*. The four continuous attributes are sepal length and width, and petal length and width.

e) Glass

This data set was downloaded from the UC Irvine Data Repository. The data consist of 214 cases containing seven classes and nine continuous attributes. The class distribution is 70 float-processed building window glasses, 17 float-processed vehicle window glasses, 76 non-float-processed building window glasses, zero non-float-processed vehicle windows, 13 containers, 9 tableware items and 29 headlamps. All attributes are continuous.

f) Sonar

This data were also downloaded from the UC Irvine Data Repository. The data set contains 208 cases of which 111 cases were obtained by bouncing sonar signals off a metal cylinder at various angles and under various conditions, and of which 97 cases were obtained by bouncing signals off of rocks under similar conditions. There are two classes, and each case is a set of 60 numbers in the range 0.0 to 1.0.

g) Wine

This data set comes from a chemical analysis of wines grown in a particular region in Italy. The data consist of 178 cases: 59 of class-1 wines, 71 of class-2 wines and 48 of class-3 wines. Each case has thirteen continuous attributes measuring the chemical properties of the wines. They are 1) Alcohol, 2) Malic acid, 3) Ash, 4) Alcalinity of ash, 5) Magnesium, 6) Total phenols, 7) Flavanoids, 8) Nonflavanoid phenols, 9) Proanthocyanins, 10) Color intensity, 11) Hue, 12) OD280/OD315 of diluted wines and 13) Proline. The data were downloaded from the UC Irvine Data Repository.

B. PERTURBATION TECHNIQUES

Different perturbation methods were used according to the type of data.

1. Categorical Data

Let p be the perturbation rate. Let $X = (x_1, x_2, \ldots, x_n)$ be the vector of categorical values and suppose that x_j takes one of the categories from $C = \{c_1, c_2, \ldots, c_q\}$. Also suppose that the empirical distribution of the categories is $(\pi_1, \pi_2, \ldots, \pi_q)$, where $\pi_1 + \pi_2 + \ldots + \pi_q = 1$. The probability π_j is obtained as the proportion of class j in the observed cases.

For each j, draw δ_j from a Bernoulli distribution with probability of success p. If $\delta_j = 1$, change the x_j to a category drawn from the distribution $P(x)$, where $P(x)$ is defined as follows:

Supposing that $x_j = c_k$,

$$P(x)=\begin{cases} \dfrac{\pi_1}{1-\pi_k} & for\ x=c_1 \\ \vdots & \\ \dfrac{\pi_{k-1}}{1-\pi_k} & for\ x=c_{k-1} \\ \dfrac{\pi_{k+1}}{1-\pi_k} & for\ x=c_{k+1} \\ \vdots & \\ \dfrac{\pi_n}{1-\pi_k} & for\ x=c_n \end{cases}$$

2. Continuous Data

Let p be the perturbation rate and let $X = (x_1, x_2, \ldots, x_n)$ be the vector of continuous values.

For each j, draw δ_j from N(0, σ_j^2) where σ_j = (sample standard deviation of X_j) × p. Then the perturbed data are obtained as $(x_1 + \delta_1, x_2 + \delta_2, \ldots, x_n + \delta_n)$.

C. SIMULATION

1. General

For each data set, the topology change when either the Y variable or the X variables are perturbed was measured. The perturbation rates used were 0, 0.002, ..., 0.01, 0.02, 0.04, ..., 0.28, 0.30, 0.35, ..., 0.50, 0.60, ..., 0.90. For each perturbation rate, 50 simulations were conducted except with the DNA data. For the DNA data, 30 simulations were performed for each perturbation rate.

2. Programming Language

All programs used in this thesis were written as S-Plus functions. Built-in functions of S-Plus such as `tree()`, `cv.tree()` and `prune.tree()` were used in these functions to build classification trees.

3. Treatment in the Case of Continuous X Variables

Continuous data values with more than five significant digits were rounded to five significant digits to avoid an undocumented round-off error bug in the function `tree()`.

D. CONSTRUCTION OF ORIGINAL TREES

1. Selection Rule

As described in the first chapter, when a tree model is built, a large tree is grown so that its terminal nodes are as pure as possible. Then the tree is pruned to the size

estimated to be optimal. The size of the optimal tree is usually estimated by cross-validation. After the cross-validation, the smallest size corresponding to the smallest deviance is chosen as the optimal size.

When this procedure is used to construct a tree, the resulting optimal size for the same grown tree could be different depending on the random number seeds used to split the data set in the cross-validations. The difference in size causes a difference in the topology between two pruned trees obtained from the same grown tree. To make the difference in the topology due to the cross-validation as small as possible, the most common size of the pruned tree was chosen for each data set by using the following method: For each grown tree, 30 to 500 cross-validations were conducted. Then, the size which most often gave the smallest deviance was chosen as the size for the optimal tree.

2. Original Trees for Each Data Set

The following table shows the summary of the original tree for each data set.

Data	Nodes	Frequency	Misclass. Rate	Misclass. Ratio	Depth
Digits1	29	115/200	0.2420	121/ 500	7
Digits2	10	-	0.0000	0/ 500	5
DNA	11	30/ 30	0.0350	35/1000	5
Diabetes	7	97/200	0.2279	175/ 768	4
Glass	5	25/ 50	0.3037	65/ 214	4
Iris	4	362/500	0.0267	4/ 150	4
Sonar	2	67/100	0.2404	50/ 208	2
Wine	7	218/500	0.0169	3/ 178	4

"Nodes" shows the number of terminal nodes chosen for the "best-sized" tree, and "frequency" shows how many times cross-validation produced a tree of that size. "Misclass. Rate" shows the misclassification rate for the "best-sized" tree and "Misclass. ratio" shows the number of cases misclassified by that tree. "Depth" shows the depth of the "best-sized" tree. For the Digits2 data, cross-validations were not conducted, since this data originally had no error and thus the grown tree had the best size.

Table 2: Original Trees for Each Data set

E. SIMULATION RESULTS AND DISCUSSION

The effects on the topology of perturbing the Y or the X variables are described below.

1. Perturbation of Y

Figures 7 and 8 show the total difference in topology and misclassification rate when the Y variable is perturbed. In each plot, circles indicate the total difference in topology and the solid line with circles indicates the misclassification rate.

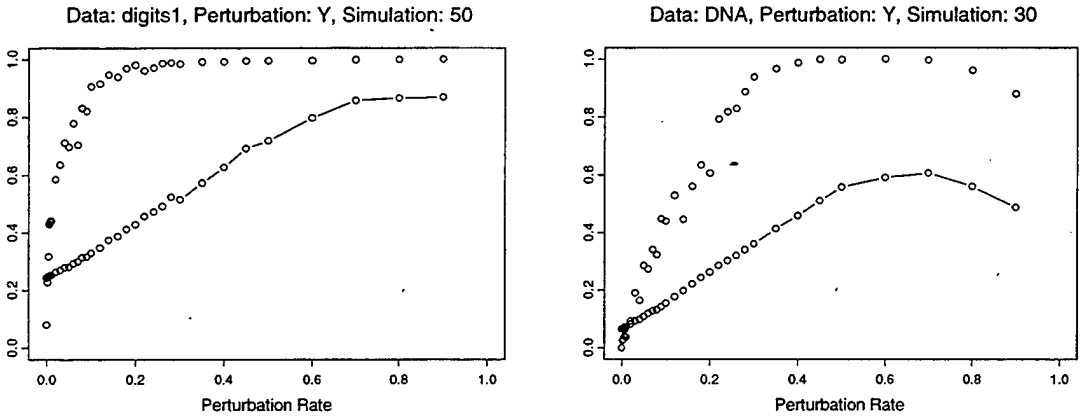
In all cases, the total difference in topology increases as the perturbation rate increases. Similarly, misclassification rates increase as perturbation rates increase for small perturbation rates. However, when the perturbation is large, misclassification rates sometimes go down.

In the case of the Diabetes and the Sonar data sets, the misclassification rates increase as the perturbation rates increase when the rates are smaller than 0.5; then they decrease as the perturbation rates continue to increase. The misclassification rates are almost symmetric about perturbation rates 0.5. The reason is that these data sets have binary response variables. So, when a case is perturbed with rate p, its class label is changed to the other class label with probability p. For instance, if we had a data set with response variable Y whose values were classes A and B, and if we perturbed Y with a very high rate, then almost all class A cases would change to B and almost all class B cases would change to A. Thus, this perturbed data set would be the same as the original one, except that each Y element would have the opposite class label in most all cases. Thus,

we would expect the misclassification rate of the tree constructed using this perturbed data to be almost equal to the original one.

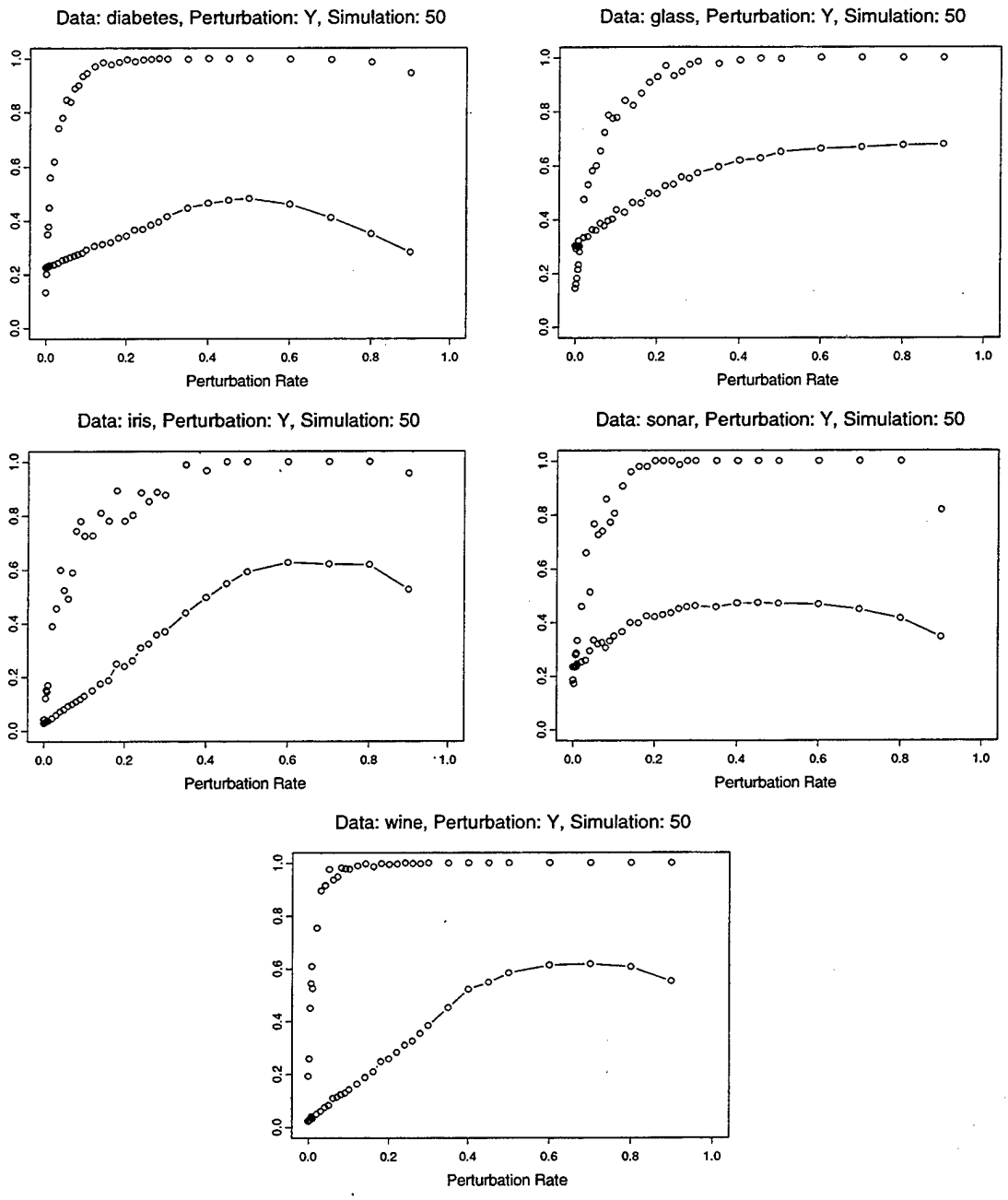
Furthermore, in the data sets with small numbers of classes, such as DNA, Iris and Wine, the misclassification rates increase at first and start to decrease for the large perturbation rates. It is plausible that something analogous to the case of binary variables occurs when the number of classes is small.

Although the behaviors of topology change and misclassification rate change are similar, the speed of change is quite different. The total difference in topology reaches 1 (which means the splitting rule is different at the root node) very quickly. On the other hand, the misclassification error increases roughly linearly.



In each plot, circles indicate the total difference in topology, and the solid line with circles indicates the misclassification rate.

Figure 7: Total Difference in Topology between Original Tree and Perturbed Tree, and Misclassification Rate When Perturbing *Y* (1). Total difference in topology and the misclassification rate increase as the perturbation rate increases in both data sets.



In each plot, circles indicate the total difference in topology, and the solid line with circles indicates the misclassification rate.

Figure 8: Total Difference in Topology between Original Tree and Perturbed Tree, and Misclassification Rate When Perturbing *Y* (2). Total difference in topology and the misclassification rate increase as the perturbation rate increases in each data set. The misclassification rates go down for large perturbation rates in the data sets with small number of classes, such as Diabetes, Iris and Sonar.

After obtaining the general results described above, some data sets were examined more closely to see how the perturbed trees were different from the original tree. An example from these observations is given in Figure 9. The tree on the left side was built using the original Diabetes data set, while the tree on the right side was constructed with the same data set where the Y variable was perturbed with perturbation rate 0.01. Although the perturbation rate is small, these two trees have different splitting rules at the root node, which implies the total difference in topology is 1 between two trees. On the other hand, the misclassification rates are not too different (0.2279 for the original tree and 0.2096 for the perturbed tree). This example suggests that the tree topology can be very different when the data are measured with error from when it is not, even though the error may be small and the misclassification rates almost the same.

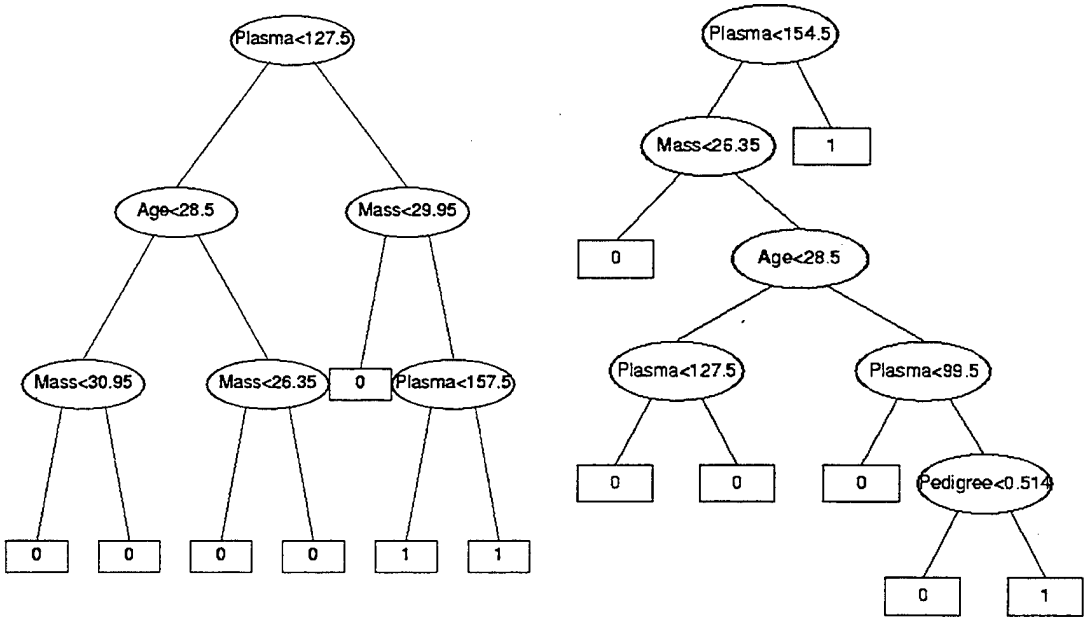


Figure 9: Original Tree (left) and Perturbed Tree (right) Using Diabetes Data. The perturbed tree is built with the data in which the Y variable is perturbed with the perturbation rate 0.01. These two trees have different splitting rules at the root node although the perturbation rate is very small.

2. Perturbation of X

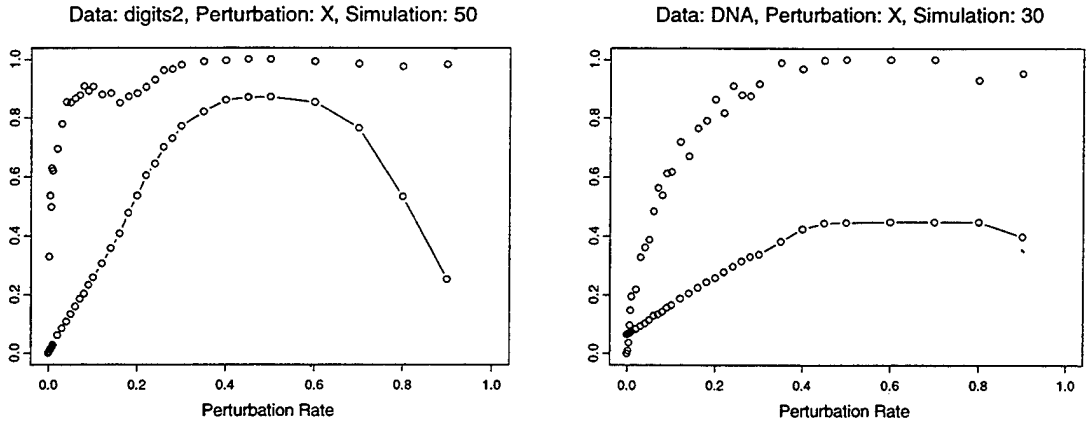
Figures 10 and 11 show the total difference in the topology and the misclassification rate when the *X* variables were perturbed. As in the previous section, circles indicate the total difference in the topology and the solid line with circles indicates the misclassification rate.

The results are similar to the case of perturbing the *Y* variables. In all cases, the total differences in topology increase as the perturbation rates increase. Unlike other data sets, the change in topology stays at almost 0.4 for perturbation rates smaller than 0.2 in the Iris data. In this data set, the original tree has four terminal nodes and its total depth is four. From this structure, it is supposed that the splitting rules may be the same in the first two levels from the top for most of these small perturbation rates.

The misclassification rates also increase as perturbation rates increase except in the case of the Digits data. In this case, the misclassification rate increases at first, then decreases for perturbation rates greater than 0.5. As we saw in the Diabetes data when perturbing the *Y* variable, the misclassification rate is symmetric about the rate 0.5. The reason is presumed to be that this data set has binary *X* variables.

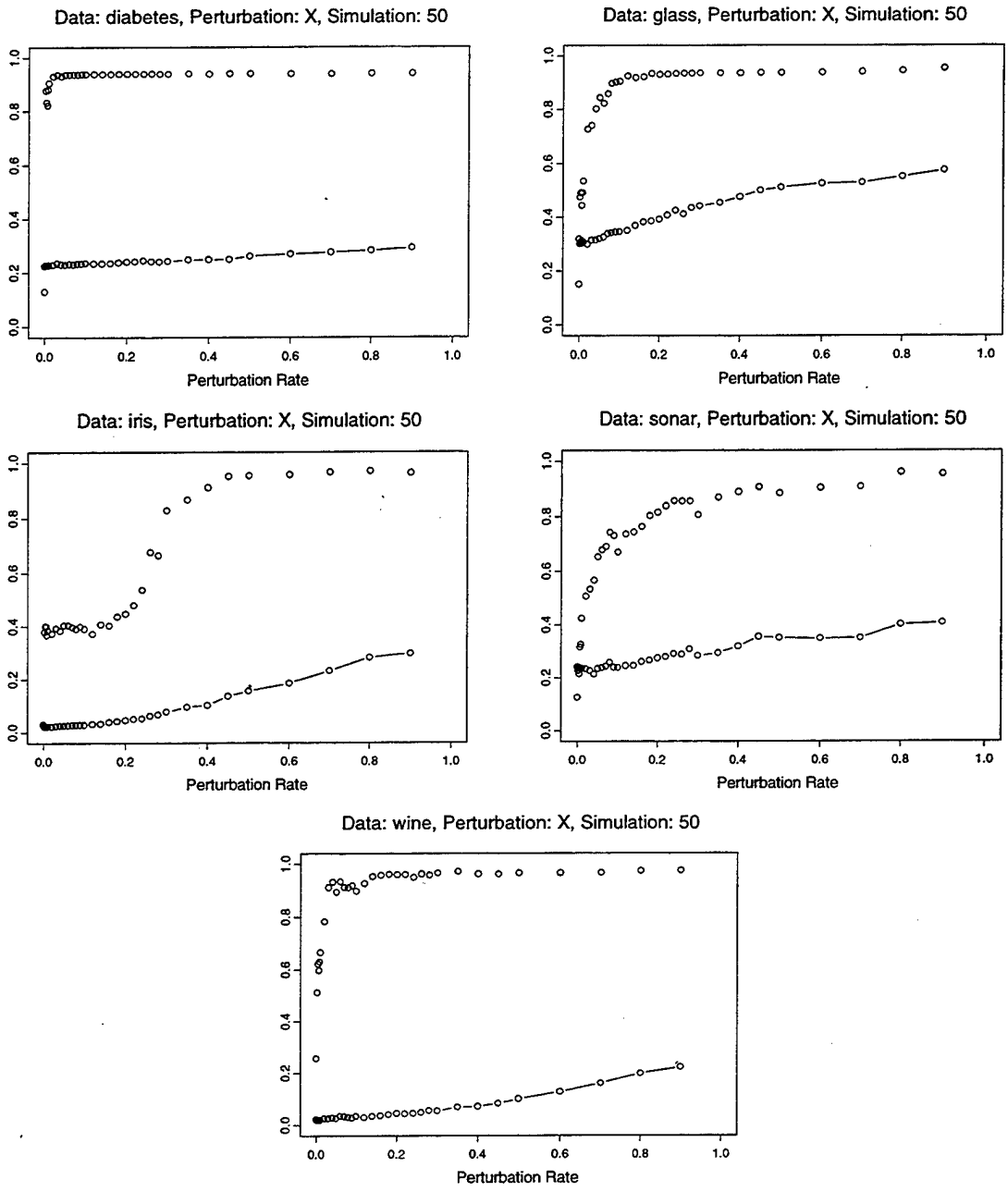
The speed of change is different as it was in the case of perturbing the *Y* variables. The total difference in topology increases approximately exponentially and approaches 1 except in the case of the Iris data. As described above, in this data, the total difference does not increase much for the small perturbation rates and then increases quickly for

rates greater than 0.2. In contrast to the difference in the topology, the misclassification rate increases almost linearly as in the case of the perturbation of the Y variables.



In each plot, circles indicate the total difference in topology, and the solid line with circles indicates the misclassification rate.

Figure 10: Total Difference in Topology between Original Tree and Perturbed Tree, and Misclassification Rate When Perturbing X (1). Both data have the categorical X variables. The total difference in topology and the misclassification rate increase as the perturbation rate increases when the perturbation rate is small in each data set. The misclassification rate decreases for large perturbation rates in Digits2.



In each plot, circles indicate the total difference in topology, and the solid line with circles indicates the misclassification rate.

Figure 11: Total Difference in Topology between Original Tree and Perturbed Tree, and Misclassification Rate When Perturbing X (2). All data sets have the continuous X variables. The total difference in topology and the misclassification rate increase as the perturbation rate increases in each data set.

IV. CONCLUSIONS

In this thesis we proposed a measure to compare the topology of two classification trees. Trees were grown using a number of well-known data sets and pruned back to the "best size" using cross-validation. Then either the Y or X variables were slightly perturbed and trees grown from this perturbed data set.

It was observed that the total difference in topology was not as large in the Iris data as in other data sets when continuous X variables were perturbed at a small rate. This suggests that when the classes are readily separable, as in this data set, small measurement errors in continuous X variables do not affect the tree topology very much.

However, in most data sets used in this thesis, the total difference in the topology exceeded 0.8 for a perturbation rate of 0.2 when perturbing either the Y or the X variables, which shows that the topology changes much even for moderate errors in the variables. One example was shown in the previous chapter. This result suggests that classification trees are not suitable for making policy decisions when people also pay attention to the splitting rules, not just to the misclassification rates.

Turning to the robustness of the tree, the total difference in the topology increased at a much higher speed than the misclassification rate did in all cases. This implies that tree models are more robust in terms of the misclassification rate than in terms of the topology.

A. FURTHER RESEARCH

In this thesis, we did not consider differences in structure (size or depth) between two trees. The change in the size or the depth given in Appendix A, however, indicates that the structure also changes when the data are perturbed. These changes are not so small that we can ignore them easily, and further research is needed in this area. When taking the difference in structure into account, defining a sort of distance between two trees with different structures and combining it with the measure of difference defined in this thesis is difficult. Miclet (1986) suggests one way to define the distance between two trees that have different structures.

As mentioned in Chapter II, when the X variable is continuous and it is perturbed, the total difference in topology could be quite high even if two trees are very similar in their topology. Thus more work is needed to improve the definition of the measure in that case.

In the process of finding the original trees, we found that the optimal size could depend on how the data were split into subsets in the cross-validation, even though cross-validations were conducted on the same grown tree. More careful consideration may be needed on how to reduce the effect of cross-validation on the difference in topology.

As a final note, in this thesis, we used data in which all of the X variables are the same type. It might be interesting to consider the case in which the X variables are both categorical and continuous.

APPENDIX A. CHANGES IN STRUCTURE

The size and depth were recorded during our simulation experiments, and these two factors convey information about the structure. In this appendix we look at how the perturbation of the data changes the structure of trees.

For reference, the number of terminal nodes and the depth of the original tree for each data are shown again in the table below.

Data	Number of Terminal nodes	Depth
Digits1	29	7
Digits2	10	5
DNA	11	5
Diabetes	7	4
Glass	5	4
Iris	4	4
Sonar	2	2
Wine	7	4

Table A1: Extract from Table 2

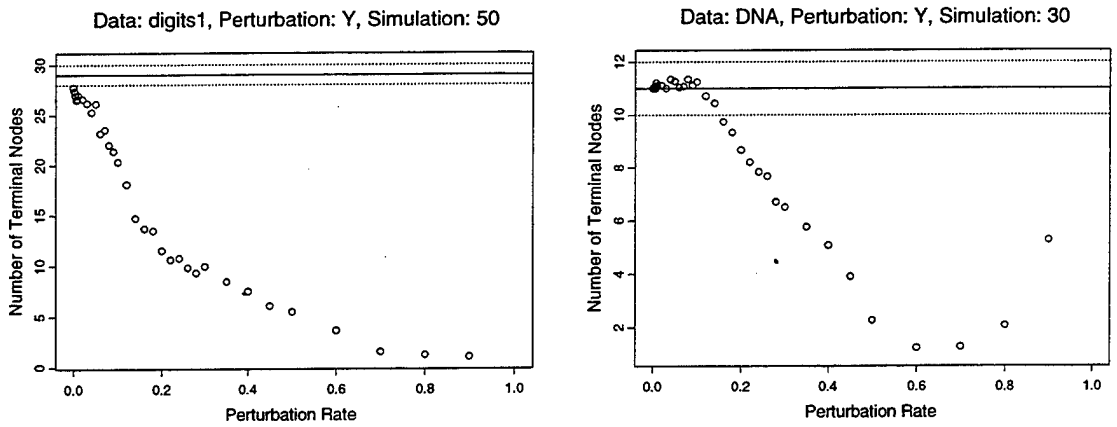
A. PERTURBATION OF *Y*

1. Number of Terminal Nodes

Figures A-1 and A-2 show the changes in the size of the tree when the *Y* variables are perturbed. In each plot, the solid line indicates the number of nodes of the original tree, and the dotted lines are drawn one node above and one below it.

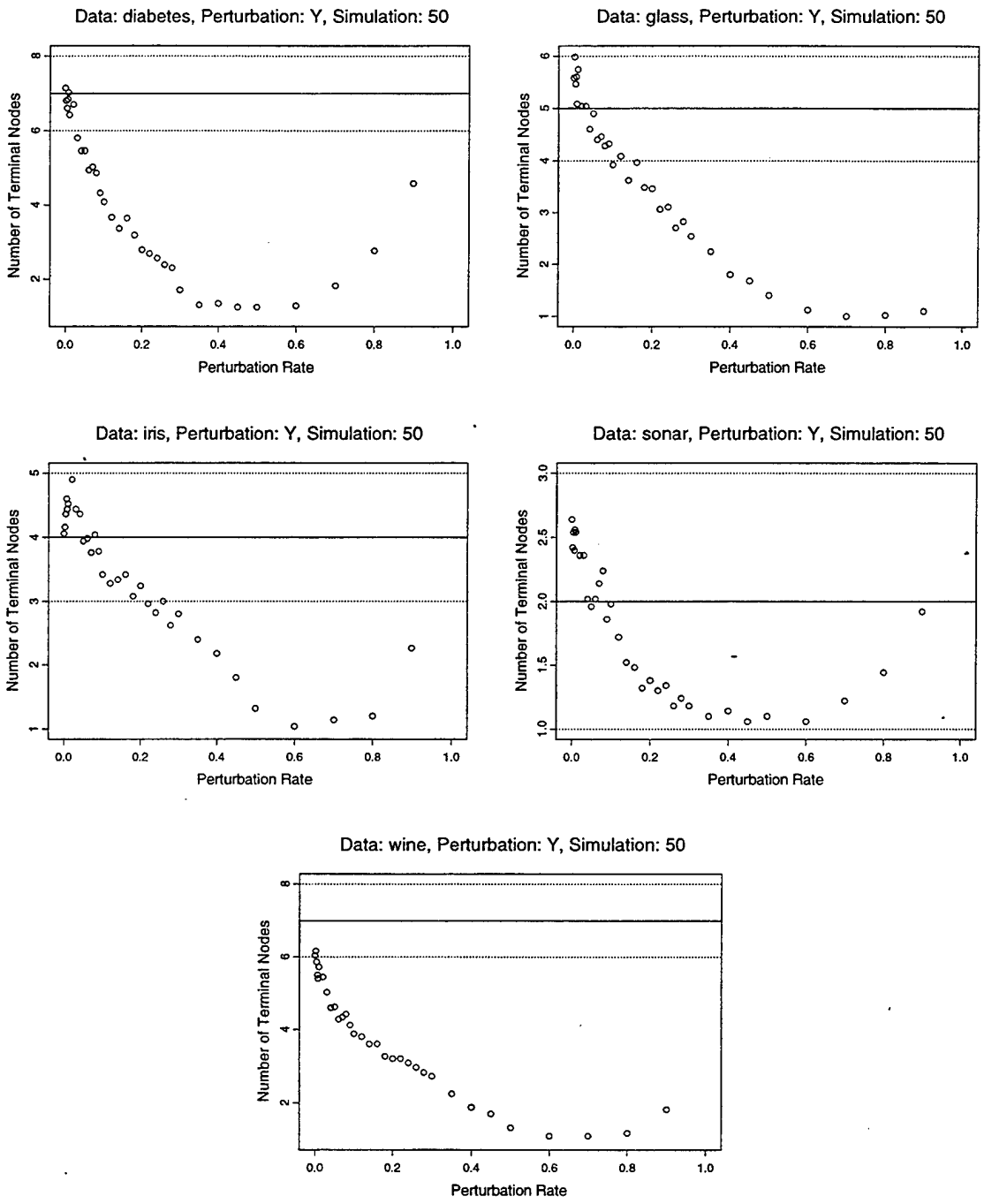
In most cases, the size decreases and approaches 1, which represents a single node tree, as the perturbation rate increases. For the binary response data, such as Diabetes and Sonar, the size decreases at first and increases for the perturbation rates greater than 0.5.

For small perturbation rates less than 0.2, the changes in size are roughly within 1 except for the Digits1, the Diabetes and the Wine data.



In each plot, the solid line indicates the number of nodes of the original tree, and the dotted lines are drawn one node above and one below it.

Figure A-1: Number of Terminal Nodes When Perturbing *Y* (1). The size decreases as the perturbation rate increases when the perturbation rates are small.



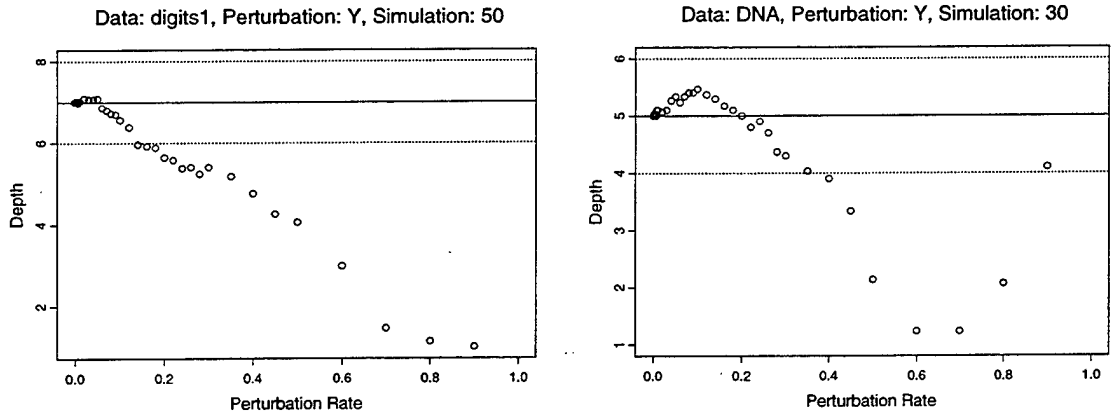
In each plot, the solid line indicates the number of nodes of the original tree, and the dotted lines are drawn one node above and one below it.

Figure A-2: Number of Terminal Nodes When Perturbing *Y* (2). The size decreases and approaches 1 as the perturbation rate increases. For the data sets with the binary *Y* variables, such as Diabetes and Sonar, the size increases for the perturbation rates greater than 0.5.

2. Depth

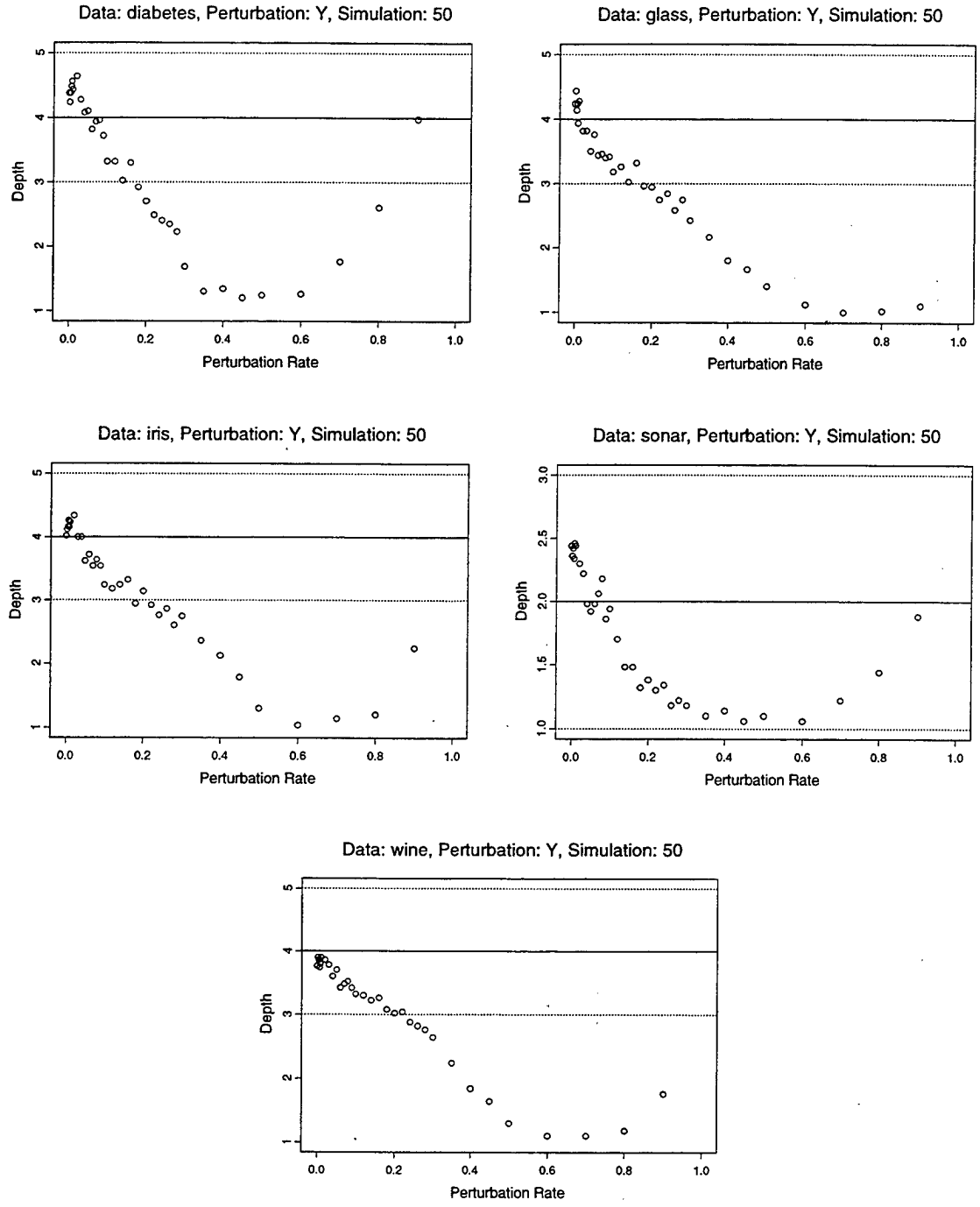
From Figures A–3 and A-4, we can see how the depth changes when perturbing the *Y* variables. The lines are drawn in the same way as in the previous section.

As in the case of size, the depth also decreases as the perturbation rate increases and approaches 1. For large perturbation rates (greater than 0.2), the perturbed tree is at least one level smaller than the original tree in the total depth except for the Sonar data. For the binary response data sets (the Diabetes and Sonar), the depth decreases and approaches 1 at first, and then increases again for the perturbation rates greater than 0.5. In these cases, depth is almost symmetric about the rate 0.5.



In each plot, the solid line indicates the depth of the original tree, and the dotted lines are drawn one depth above and one below it.

Figure A-3: Depth When Perturbing *Y* (1). The depth decreases as the perturbation rate increases and approaches 1.



In each plot, the solid line indicates the depth of the original tree, and the dotted lines are drawn one depth above and one below it.

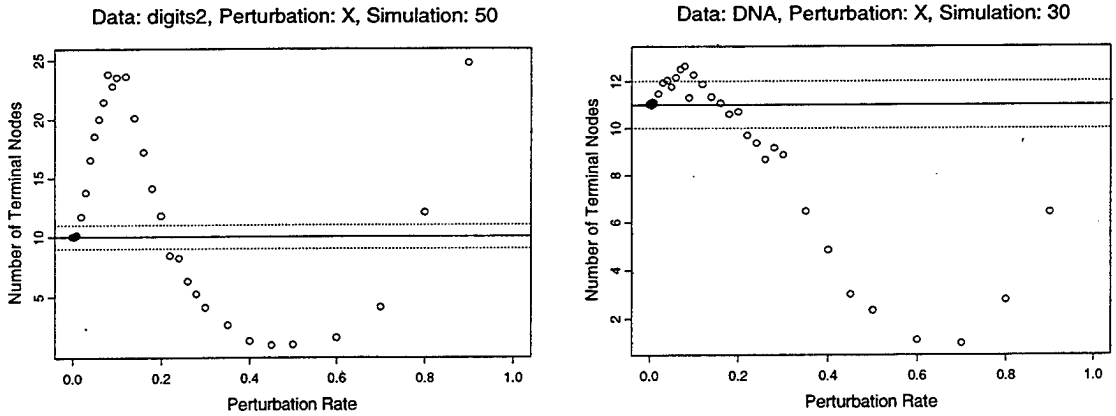
Figure A-4: Depth When Perturbing *Y* (2). The depth decreases as the perturbation rate increases, and approaches 1. In the case of the data sets with the binary *Y* variable (Diabetes and Sonar), the depth is almost symmetric about the rate 0.5.

B. PERTURBATION OF *X*

1. Number of Terminal Nodes

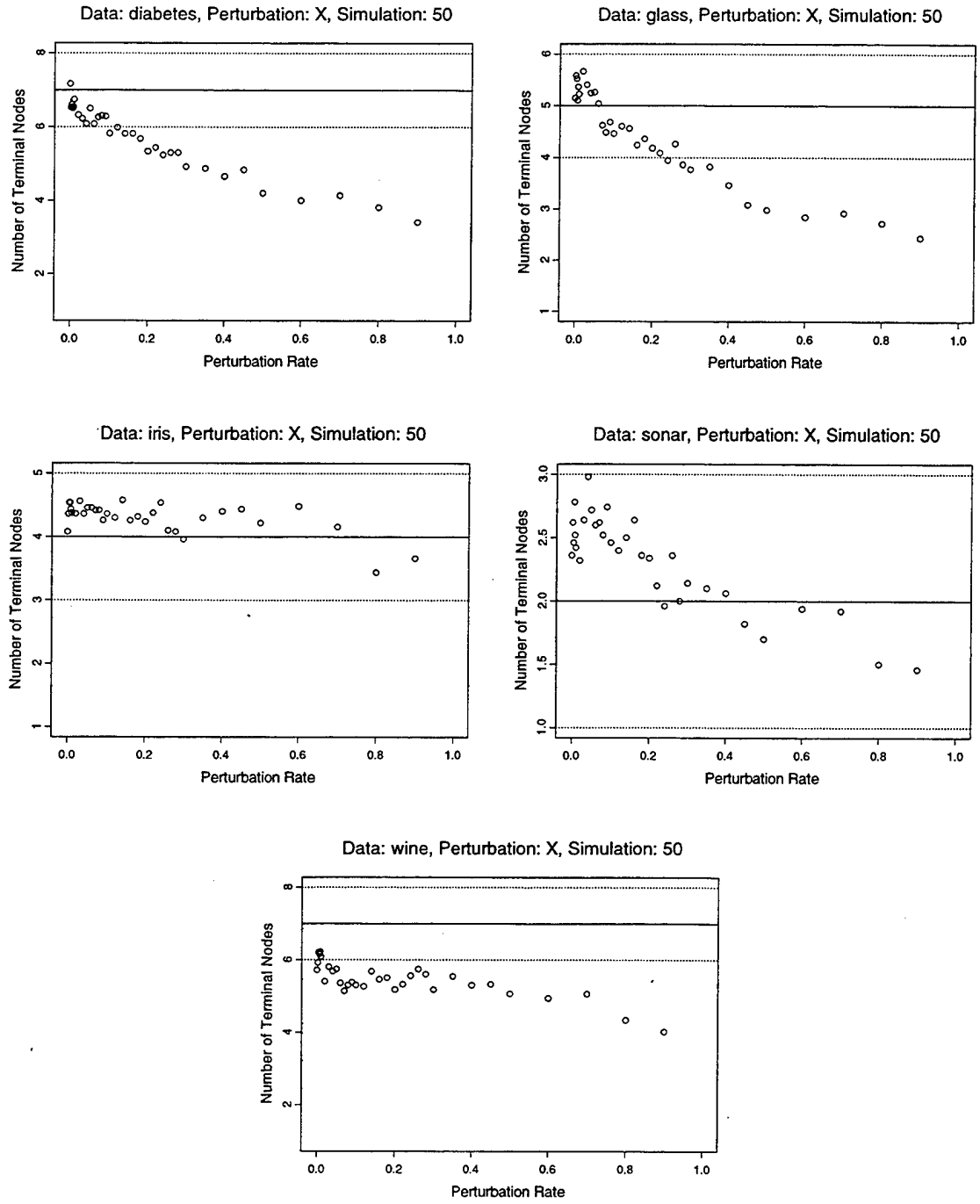
Figures A-5 and A-6 show the changes in tree size when the *X* variables are perturbed. The lines are drawn in the same way as in the previous sections in each plot.

The size decreases as the perturbation rate increases in most sets of data. Among all sets of data, the Digits2 and the DNA data sets behave differently from the other data sets: the sizes increase for the perturbation rates 0 to about 0.1, start to decrease and increase again for large perturbation rates. In the Digits2 data, the size is almost symmetric about the rate 0.5 for perturbation rates greater than 0.1. Turning to the Iris and the Wine data, the size does not reduce much compared with other data sets. The changes in the size are about 1 for the Iris data and about 2 for the Glass data for perturbation rates smaller than 0.6. In contrast, the size decreases almost linearly as the perturbation rate increases in the Diabetes and the Glass data.



In each plot, the solid line indicates the number of nodes of the original tree, and the dotted lines are drawn one node above and one below it.

Figure A–5: Number of Terminal Nodes When Perturbing *X* (1). Both data sets have the categorical *X* variables. The size increases for the small perturbation rates, decreases and increases again for large perturbation rates.



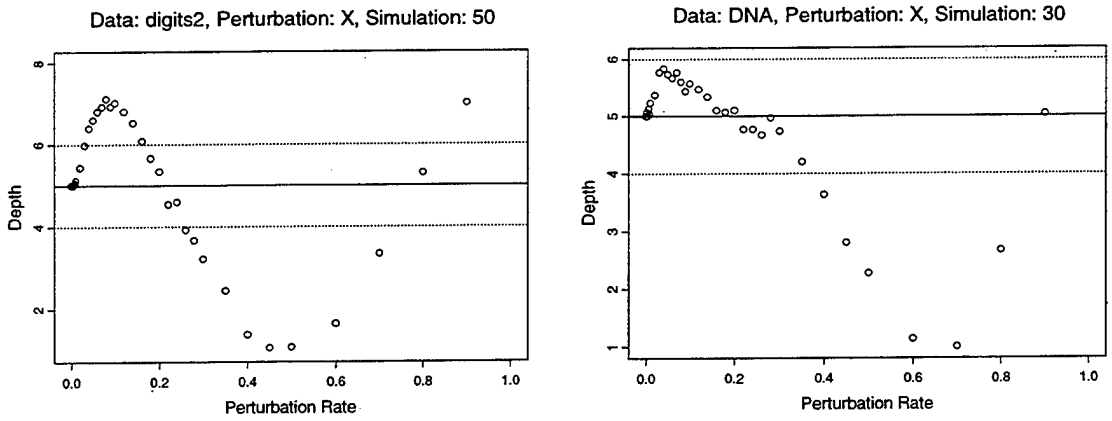
In each plot, the solid line indicates the number of nodes of the original tree, and the dotted lines are drawn one node above and one below it.

Figure A–6: Number of Terminal Nodes When Perturbing X (2). The X variables are continuous in all data sets. The size decreases as the perturbation rate increases. The size does not reduce much in the Iris and Wine data sets.

2. Depth

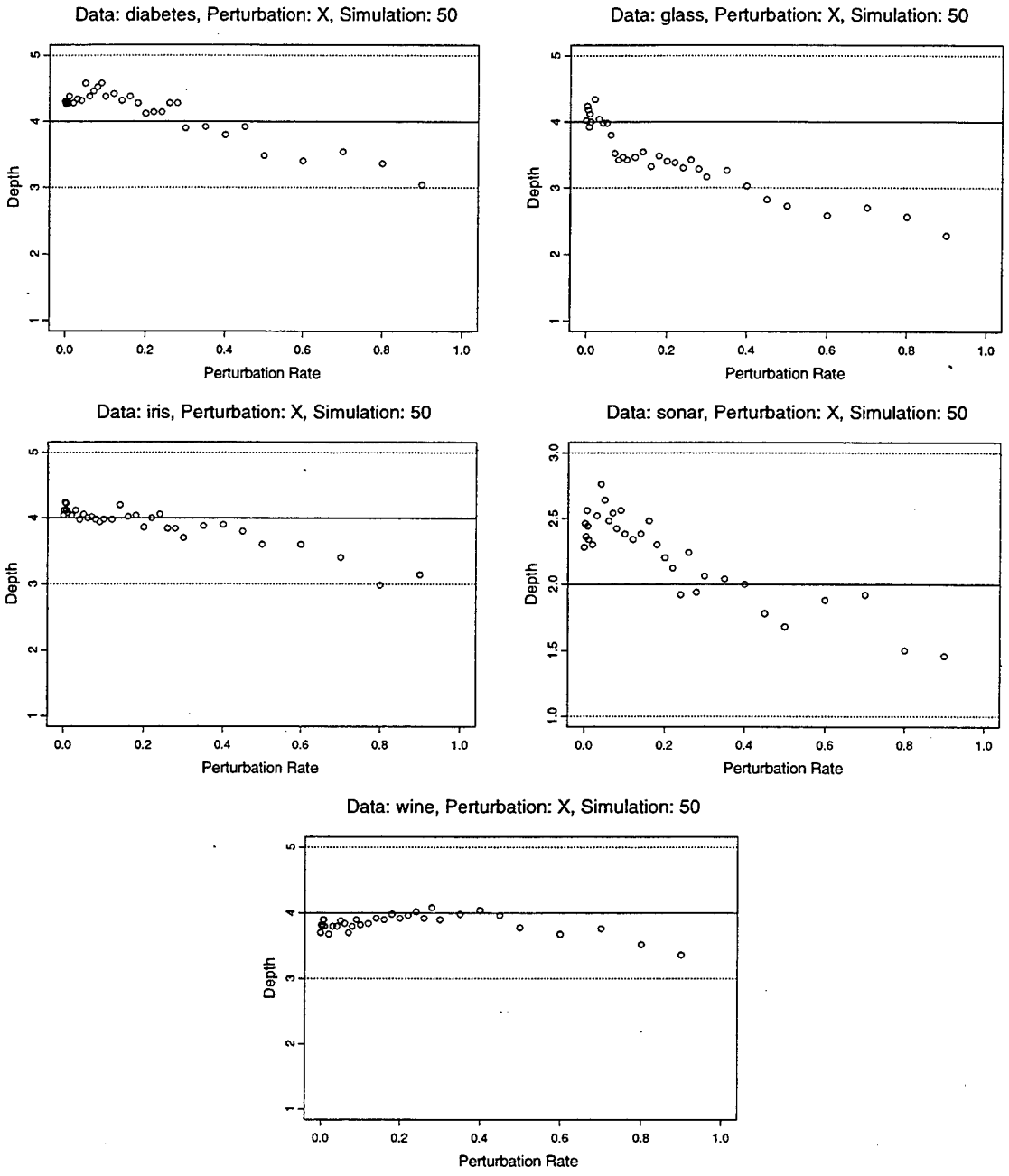
Figures A-7 and A-8 show how the depth changes as the X variables are perturbed. The depth decreases as the perturbation rate increases for most data sets. The solid line and the dot lines are drawn as in the previous sections.

For the Digits2 or the DNA data sets, the behavior is very similar to that of the size. Unlike other data sets, the depth stays almost constant for the rate less than 0.5 in the Iris and the Wine data. In data sets with continuous X variables, the changes in the depth are approximately within one unit from those of the original data sets, except for the Glass data.



In each plot, the solid line indicates the depth of the original tree, and the dotted lines are drawn one depth above and one unit below it.

Figure A–7: Depth When Perturbing X (1). The X variables are categorical in both data sets. The depth increases first, decreases and approaches 1; increases again for large perturbation rates.



In each plot, the solid line indicates the depth of the original tree, and the dotted lines are drawn one depth above and one unit below it.

Figure A–8: Depth When Perturbing X (2). All data sets have continuous X variables. The depth stays constant for small perturbation rates in the Iris and Wine. The changes in the depth are approximately within one unit from those of the original data sets in most cases.

THIS PAGE INTENTIONALLY BLANK

APPENDIX B. S-PLUS CODE

This appendix contains the S-Plus code for functions used to simulate the change in tree topology in this thesis. The functions used for perturbing the *Y* variables (`tree.sim2()`) and the *X* variables (`tree.sim.x2()`) are shown in section A and B, respectively. These functions are essentially the same, with the exception of the parts for perturbing variables and computing the change in topology.

The function called `chy()` used to perturb categorical data in the functions described above is given in section C.

A. FUNCTION FOR PERTURBING *Y*

```
function(object, nom.sim, perturb.rate, is.x.cat = T, org.tree = NULL,
...)
{
#
# tree.sim2
# June 30, 1999
#
# Program to simulate the change in tree topology when the response
#    variable is perturbed.  The response variable must be in column 1
#    in data set.
#
# Input:
#     object       : name of data
#     nom.sim      : the number of simulation
#     perturb.rate: the size of perturbation
#     is.x.cat     : T if X variable is categorical, F if continuous
#     org.tree     : name of original tree
#
# Output:
#     perturb      : perturbation rate
#     topology     : total change in tree topology (mean)
#     top.std      : total change in tree topology (standard deviation)
#     misclass     : misclassification rate (mean)
#     m.std        : misclassification rate (standard deviation)
#     depth        : depth (mean)
#     d.std        : depth (standard deviation)
#     nodes        : number of terminal nodes (mean)
#     n.std        : number of terminal nodes (standard deviation)
#
      cat("call: ", "\n")
      cat(list(match.call()))
```

```
      cat("\n")
      per <- sort(perturb.rate)     #
#
      object <- data.frame(object)
      names(object)[1] <- "Y"
      assign("org.data", data.frame(object), frame = 1)

#
# Make the type of the response variable factor so that the function
#    tree() will build a classification tree.  If X is categorical, make
#    their type factor.  If continuous, round them to 5 significant
#    digits.
#

      org.data[, 1] <- as.factor(org.data[, 1])
      if(is.x.cat == T) {
            for(i in 2:ncol(org.data))
                org.data[, i] <- as.factor(org.data[, i])
      }
      else {
            for(i in 2:ncol(org.data))
                org.data[, i] <- signif(org.data[, i], 5)
      }
      assign("org.data", org.data, frame = 1)   #

#
# Construct a tree model for the original data if the original tree is
#    not given as input.
#

      if(!inherits(org.tree, "tree")) {
            org.tree <- tree(Y ~ ., data = org.data)
            org.cv <- cv.tree(org.tree, FUN = prune.tree)
            nom.node <- min(org.cv$size[org.cv$dev == min(org.cv$dev)])
            org.prune <- prune.tree(org.tree, best = nom.node)    #
            org.tree <- org.prune
      }
      else {
            nom.node <- summary(org.tree)$size
      }
      total.nodes <- max(as.numeric(dimnames(org.tree$frame)[[1]]))
      depth <- floor(log(total.nodes, base = 2)) + 1  #
#
# Create a matrix for the original tree.
#
# The information in this matrix is
#     node: nodes that exist in the tree (column 1)
#     var : variables (column 2)
#     split.left, split.right: split rules in the nodes (column 3 & 4)
#
      org <- matrix(0, 2^depth - 1, 4)
                              # make the original tree to be complete
      dimnames(org) <- list(NULL, c("node", "var", "split.left",
            "split.right"))
      for(i in 1:nrow(org)) {
            org[i, 1] <- as.numeric(dimnames(org.tree$frame)[[1]][
                  as.numeric(dimnames(org.tree$frame)[[1]]) == i])
```

```
            org[i, 2] <-org.tree$frame[as.numeric(dimnames(
                  org.tree$frame)[[1]]) == i,  ]$var
            org[i, (3:4)] <- org.tree$frame[as.numeric(dimnames(
                  org.tree$frame)[[1]]) == i,  ]$splits
      }
#
# Prepare vectors to store the results of simulations.
#
      measure.topology <- rep(0, length(per))
                                    # measure of change in topology
      misclass.rate <- rep(0, length(per))# misclassification rate
      mean.depth <- rep(0, length(per))    # depth
      mean.nodes <- rep(0, length(per))    # number of terminal nodes
      top.std <- rep(0, length(per))       # std of topology
      misclass.std <- rep(0, length(per)) # std of misclass rate
      depth.std <- rep(0, length(per))     # std of depth
      nodes.std <- rep(0, length(per))     # std of terminal nodes #
#
      used.seed <- array(-1, c(length(per), 12))
                              # Random number seed used for the first
                              #  simulation for each perturbation rate
#
# Divide the data set into the Y and the X to prepare for perturbing
#    the Y.  When the X is categorical, make their type factor.
#
      x <- org.data[, -1]      # X variable
      newY <- org.data[, 1]
      new.data <- data.frame(cbind(newY, x))
      names(new.data)[1] <- "newY"
      assign("new.data", data.frame(new.data), frame = 1)
      new.data[, 1] <- as.factor(new.data[, 1])
      if(is.x.cat == T) {
            for(i in 2:ncol(new.data))
                  new.data[, i] <- as.factor(new.data[, i])
      }
#
      for(perturb in 1:length(per)) {
            first.seed <- .Random.seed
            cat("Perturb Rate: ", per[perturb], "\t")
            cat(": seed : ", first.seed, "\n")
            used.seed[perturb,  ] <- first.seed #
#
# Create vectors to store the result of each simulation.
#
            mot.vec <- rep(0, nom.sim)
                              # Measure of change in topology
            cnode.vec <- rep(0, nom.sim)
                              # Nodes where splitting rule is changed
            misclass.vec <- rep(0, nom.sim)      # Misclassification rate
            depth.vec <- rep(0, nom.sim)  # Depth
            nodes.vec <- rep(0, nom.sim)  # Number of terminal nodes
#
            for(case in 1:nom.sim) {
#
# Perturb Y variable using the function chy(), ...
#
                  new.data[, 1] <- chy(newY, per[perturb])  #
```

```
                    assign("new.data", new.data, frame = 1)
                    new.data[, 1] <- as.factor(new.data[, 1]) #
#
# ...and construct a tree using this newY.
#
                    new.data.tree <- tree(newY ~ ., data = new.data)
                    new.data.cv <- cv.tree(new.data.tree,
                          FUN = prune.tree)
                    new.nodes <- min(new.data.cv$size[new.data.cv$dev
                           == min(new.data.cv$dev)])
                    cmp.tree <- prune.tree(new.data.tree,
                          best = new.nodes) #
#
# Compute the misclassification rate and ...
#
                    if(new.nodes > 1) {
                          misclass.vec[case] <-
                            summary(cmp.tree)$misclass[1]/
                            summary(cmp.tree)$misclass[2]
                    }
                    else {
                          misclass.vec[case] <-
                            1 - max(table(new.data[, 1])/nrow(new.data))
                    }
#
# ... the depth of the perturbed tree.
#
                    new.depth <- floor(log(max(as.numeric(dimnames(
                          cmp.tree$frame)[[1]])), base = 2)) + 1
                    depth.vec[case] <- new.depth
                    nodes.vec[case] <- new.nodes  #
#
# Create a matrix for the perturbed tree.  This matrix has five columns.
#    The number of rows is determined as follows.  If the maximum number
#    of nodes (called N) of the perturbed tree is greater than that of
#    the original one (assuming it's complete), the number of rows = N.
#    If not, the number of rows is the same as that of matrix org.
#
# The information in this matrix is
#     node: nodes that exist in the tree (column 1)
#     var : variables (column 2)
#     split.left, split.right: split rules in the nodes (column 3 & 4)
#     changed: flag to tell if node is changed from the original tree,
#           and this flag is also used to check if the node has already
#           been checked (column 5)
#
                    if(max(as.numeric(dimnames(cmp.tree$frame)[[1]])) >=
                          nrow(org)) {
                          mat.row <-max(as.numeric(dimnames(
                            cmp.tree$frame)[[1]]))
                    }
                    else {
                          mat.row <- nrow(org)
                    }
                    cmp <- matrix(0, mat.row, 5)
                    dimnames(cmp) <- list(NULL, c("node", "var",
                          "split.left", "split.right", "checked"))
```

```
                    for(i in 1:nrow(cmp)) {
                          cmp[i, 1] <-
                            as.numeric(dimnames(cmp.tree$frame)[[1]][
                            as.numeric(dimnames(cmp.tree$frame)[[1]])
                            == i])
                          cmp[i, 2] <-
                            cmp.tree$frame[as.numeric(dimnames(
                            cmp.tree$frame)[[1]]) == i,  ]$var
                          cmp[i, (3:4)] <-
                            cmp.tree$frame[as.numeric(dimnames(
                            cmp.tree$frame)[[1]]) == i,  ]$splits
                    }
#
# See each node to check if the variable and the splitting value is
#    changed and put in one of the following flags.
#
#     1: No change
#     2: Nodes having the number which is greater than the total
#        number of nodes in the original tree
#     3: Variable is changed
#     4: Split rule is changed.  Variable is the same.
#

#
# Put 2 if the node number is greater than the maximum node number of
#    the original tree.  These nodes will not be checked later.
#
                    if(nrow(cmp) > nrow(org)) {
                          for(j in (nrow(org) + 1):nrow(cmp)) {
                            cmp[j, 5] <- 2
                          }
                    }
                    for(i in 1:nrow(org)) {
#
                          if(cmp[i, 5] != "1" && cmp[i, 5] != "2") {
#
# Put 1 if the splitting rule is the same as that of the original tree.
#
                            if(cmp[i, 2] == org[i, 2]) {
                              if(cmp[i, 3] == org[i, 3]) {
                                cmp[i, 5] <- 1
                              }
                              else {
#
# Put "4" if the variable is the same but the splitting value is
#    different...
#
                                cmp[i, 5] <- 4    #
#
# ... and check child nodes and put "1" (which means "checked,") in
#    order not to check them later.
#
                                depth1 <- floor(log(i, base = 2)) + 1
                                if(depth > depth1) {
                                  for(k in 1:(depth - depth1)) {
                                    for(j in 1:2^k) {
                                      cmp[(i * 2^k + j - 1), 5] <- 1
```

```
                                    }
                                  }
                                }
                              }
                            }
                            else {
#
# Put "3" if the variable used is different from the original tree...
#
                              cmp[i, 5] <- 3      #
#
# ... and check child nodes and put "1" (,which means "checked,") not to
# check them later.
#
                              depth1 <- floor(log(i, base = 2)) + 1
                              if(depth > depth1) {
                                for(k in 1:(depth - depth1)) {
                                  for(j in 1:2^k) {
                                    cmp[(i * 2^k + j - 1), 5] <- 1
                                  }
                                }
                              }
                            }
                          }
#
# Calculate the total change in tree topology.
#
                          mot <- 1

#
# If there is no change, total change in topology = 0.
#
                          cnode <- NULL
                          if(length(cmp[, 5][cmp[, 5] == "1"]) == nrow(cmp)) {
                                mot <- 0
                          }
#
# If there is any change in splitting rule, pick up the node numbers
#    where the change is observed ...
#
                          else {
                                cnode <- as.numeric(cmp[as.numeric(cmp[, 5])
                                  >= 3,   , drop = F][, 1])
                          }
                          cnode.vec[case] <- list(cnode)
#
# ... and compute the depth of these nodes.  Then, compute the total
#    change in tree topology.
#
                          if(mot != 0) {
                                cd <- floor(log(cnode, 2)) + 1
                                mot <- sum(2^(depth - cd + 1) - 1)/(2^depth - 1)
                          }
                          mot.vec[case] <- mot    #
                    }
#
```

```
# Store the result for each perturbation rate.
#
            measure.topology[perturb] <- mean(mot.vec)
            misclass.rate[perturb] <- mean(misclass.vec)
            mean.depth[perturb] <- mean(depth.vec)
            mean.nodes[perturb] <- mean(nodes.vec)
            top.std[perturb] <- sqrt(var(mot.vec))
            misclass.std[perturb] <- sqrt(var(misclass.vec))
            depth.std[perturb] <- sqrt(var(depth.vec))
            nodes.std[perturb] <- sqrt(var(nodes.vec))      #
      }

#
# Output the results.
#
      result <- data.frame(cbind(per, measure.topology, top.std,
            misclass.rate, misclass.std, mean.depth, depth.std,
            mean.nodes, nodes.std))
      names(result) <- c("perturb", "topology", "top.std", "misclass",
            "m.std", "depth", "d.std", "nodes", "n.std")
      used.seed <- cbind(per, used.seed)
      result.all <- list(call = match.call(), first.seed = used.seed,
            result = result)
      result.all
}
```

B. FUNCTION FOR PERTURBING *X*

```
function(object, nom.sim, perturb.rate, is.x.cat = T, org.tree = NULL,
...)
{
#
# tree.sim.x2
# July 1, 99
#
# This program is modified from tree.sim2 to simulate the change in tree
#    topology when the X variables are perturbed.  X can be continuous
#    or categorical.  The response variable must be in column 1
#    in data set.
#
# Input:
#     object      : name of data
#     nom.sim     : the number of simulation
#     perturb.rate: the size of perturbation
#     is.x.cat    : T if X variable is categorical, F if continuous
#     org.tree    : name of original tree
#
# Output:
#     perturb     : perturbation rate
#     topology    : total change in tree topology (mean)
#     top.std     : total change in tree topology (standard deviation)
#     misclass    : misclassification rate (mean)
#     m.std       : misclassification rate (standard deviation)
#     depth       : depth (mean)
#     d.std       : depth (standard deviation)
#     nodes       : number of terminal nodes (mean)
```

```
#     n.std       : number of terminal nodes (standard deviation)
#
      cat("\n")
      cat("call: ", "\n")
      cat(list(match.call()))
      cat("\n")
      per <- sort(perturb.rate)     #
#
      object <- data.frame(object)
      names(object)[1] <- "Y"
      assign("org.data", data.frame(object), frame = 1)

#
# Make the type of the response variable factor so that the function
#    tree() will build a classification tree.  If X is categorical, make
#    its type factor.  If continuous, round it to five significant
#    digits.
#
      org.data[, 1] <- as.factor(org.data[, 1])
      if(is.x.cat == T) {
            for(i in 2:ncol(org.data))
                  org.data[, i] <- as.factor(org.data[, i])
      }
      else {
            org.data[, -1] <- signif(org.data[, -1], 5)
      }
      assign("org.data", org.data, frame = 0)

#
# Construct a tree model for the original data if the original tree is
#    not given as input.
#
      if(!inherits(org.tree, "tree")) {
            org.tree <- tree(Y ~ ., data = org.data)
            org.cv <- cv.tree(org.tree, FUN = prune.tree)
            nom.node <- min(org.cv$size[org.cv$dev == min(org.cv$dev)])
            org.prune <- prune.tree(org.tree, best = nom.node)
            org.tree <- org.prune
      }
      else {
            nom.node <- summary(org.tree)$size
      }
      total.nodes <- max(as.numeric(dimnames(org.tree$frame)[[1]]))
      depth <- floor(log(total.nodes, base = 2)) + 1  #
#
# Create a matrix for the original tree.
#
# The information in this matrix is
#     node: nodes that exist in the tree (column 1)
#     var : variables (column 2)
#     split.left, split.right: split rules in the nodes (column 3 & 4)
#     node.class: number of cases falling in the node from each class
#                                                          (column 5 - )
#
      nom.classes <- length(table(org.data[, 1]))
```

```
                              # Number of classes in response variable
      org <- matrix(0, 2^depth - 1, 4 + nom.classes)
                              # make the original tree to be complete
      dimnames(org) <- list(NULL, c(c("node", "var", "split.left",
            "split.right"), 1:nom.classes))
      for(i in 1:nrow(org)) {
            org[i, 1] <-  as.numeric(dimnames(org.tree$frame)[[1]][
                  as.numeric(dimnames(org.tree$frame)[[1]]) == i])
            org[i, 2] <- org.tree$frame[as.numeric(dimnames(
                  org.tree$frame)[[1]]) == i,  ]$var
            org[i, (3:4)] <- org.tree$frame[as.numeric(dimnames(
                  org.tree$frame)[[1]]) == i,  ]$splits
            org[i, 5:(4 + nom.classes)] <- round(org.tree$frame[
                  as.numeric(dimnames(org.tree$frame)[[1]]) == i,
                  ]$n * org.tree$frame[as.numeric(dimnames(
                  org.tree$frame)[[1]]) == i,  ]$yprob)
      }
#
# Divide the data set into the Y and the X to prepare for perturbing
#    the Y.  When the X is categorical, make its type factor.  When
#    the X is continuous, calculate the standard deviation of each
#    attribute.
#
      x <- data.frame(org.data[, -1])      # X variable
      if(is.x.cat == F) {
            x.std <- sqrt(apply(x, 2, var))
      }
      Y <- org.data[, 1]                   # Y variable
      new.data <- data.frame(cbind(Y, x))
      names(new.data)[1] <- "Y"
      assign("new.data", data.frame(new.data), frame = 1)
      new.data[, 1] <- as.factor(new.data[, 1]) #
#
# Prepare vectors to store the results of simulations.
#
      measure.topology <- rep(0, length(per))
                                    # measure of change in topology
      misclass.rate <- rep(0, length(per))      # misclassification rate
      mean.depth <- rep(0, length(per))    # depth
      mean.nodes <- rep(0, length(per))    # number of terminal nodes
      top.std <- rep(0, length(per))       # std of measure.topology
      misclass.std <- rep(0, length(per)) # std of misclass rate
      depth.std <- rep(0, length(per))     # std of depth
      nodes.std <- rep(0, length(per))     # std of terminal nodes #
#
      used.seed <- array(-1, c(length(per), 12))
                              # Random number seed used for the first
                              #  simulation for each perturbation rate
#
      for(perturb in 1:length(per)) {
            first.seed <- .Random.seed
            cat("Perturb rate: ", per[perturb], "\t")
            cat(" seed : ", first.seed, "\n")
            used.seed[perturb,  ] <- first.seed #
#
# Create vectors to store the result of each simulation.
#
```

```
            mot.vec <- rep(0, nom.sim)
                                    # Measure of change in topology
            cnode.vec <- rep(0, nom.sim)
                                  # Nodes where splitting rule is changed
            misclass.vec <- rep(0, nom.sim)      # Misclassification rate
            depth.vec <- rep(0, nom.sim)  # Depth
            nodes.vec <- rep(0, nom.sim)  # Number of terminal nodes
#
            for(case in 1:nom.sim) {
                  newX <- data.frame(matrix(0, nrow(x), ncol(x)))
                  names(newX) <- names(x) #
#
# Perturb X variable.  When the X variable is categorical, use chy() for
#    each attribute.  When the X is continuous, use the Normal
#    distribution with mean = 0 and standard deviation = (perturbation
#    rate) * (std of the attribute) for each attribute.
#
                  if(is.x.cat == T) {
                        for(col in 1:ncol(x)) {
                          newX[, col] <- chy(x[, col], per[perturb])
                        }
                  }
                  else {
                        for(col in 1:ncol(x)) {
                          tao <- per[perturb] * x.std[col]
                          if(tao > 0) {
                            delta <- rnorm(nrow(x), 0, tao)
                            newX[, col] <- x[, col] + delta
                          }
                          else {
                            newX[, col] <- x[, col]
                          }
                        }
                  }
                  new.data[, -1] <- newX
                  assign("new.data", new.data, frame = 1)    #
#
# When the X is categorical, make the type factor.  When the X is
#    continuous, round it to five significant digits.
#
                  if(is.x.cat == T) {
                        for(i in 2:ncol(new.data)) {
                          new.data[, i] <- as.factor(new.data[, i])
                        }
                  }
                  else {
                        new.data[, -1] <- signif(new.data[, -1], 5)
                  }
                  assign("new.data", new.data, frame = 1)

#
# Construct a tree using new.data.
#
                  new.data.tree <- tree(Y ~ ., data = new.data)
                  new.data.cv <-
```

```
                        cv.tree(new.data.tree, FUN = prune.tree)
                  new.nodes <- min(new.data.cv$size[new.data.cv$dev
                        == min(new.data.cv$dev)])
                  cmp.tree <- prune.tree(new.data.tree,
                        best = new.nodes)
#
# Compute the misclassification rate and ...
#
                  if(new.nodes > 1) {
                        misclass.vec[case] <-
                              summary(cmp.tree)$misclass[1]/
                              summary(cmp.tree)$misclass[2]
                  }
                  else {
                        misclass.vec[case] <- 1 -
                              max(table(new.data[, 1])/nrow(new.data))
                  }
#
# ... the depth of the perturbed tree.
#
                  new.depth <- floor(log(max(as.numeric(dimnames(
                        cmp.tree$frame)[[1]])), base = 2)) + 1
                  depth.vec[case] <- new.depth
                  nodes.vec[case] <- new.nodes  #
#
# Create a matrix for the perturbed tree. This matrix has 6 + (number of
#    classes of the response variable) columns.
#    The number of rows is determined as follows.  If the maximum number
#    of nodes (called N) of the perturbed tree is greater than that of
#    the original one (assuming it's complete), the number of rows = N.
#    If not, the number of rows is the same as that of matrix org.
#
# The information in this matrix is
#     node: nodes that exist in the tree (column 1)
#     var : variables (column 2)
#     split.left, split.right: split rules in the nodes (column 3 & 4)
#     changed: flag to tell if the node is changed from the original
#         tree, and this flag is also used to check if the node has been
#         already checked (column 5)
#     node.class: number of cases falling in the node from each class
#                 (column 6 - )
#
                  if(max(as.numeric(dimnames(cmp.tree$frame)[[1]]))
                     >= nrow(org)) {
                        mat.row <- max(as.numeric(dimnames(
                              cmp.tree$frame)[[1]]))
                  }
                  else {
                        mat.row <- nrow(org)
                  }
                  cmp <- matrix(0, mat.row, 6 + nom.classes)
                  dimnames(cmp) <- list(NULL, c(c("node", "var",
                        "split.left", "split.right", "checked",
                        "change.spl"), 1:nom.classes))
                  for(i in 1:nrow(cmp)) {
                        cmp[i, 1] <- as.numeric(dimnames(cmp.tree$frame
                              )[[1]][as.numeric(dimnames(cmp.tree$frame)
```

```
                              [[1]]) == i])
                        cmp[i, 2] <- cmp.tree$frame[as.numeric(dimnames(
                              cmp.tree$frame)[[1]]) == i,  ]$var
                        cmp[i, (3:4)] <- cmp.tree$frame[as.numeric(
                              dimnames(cmp.tree$frame)[[1]]) == i,
                              ]$splits
                        cmp[i, 7:(6 + nom.classes)] <-
                              round(cmp.tree$frame[as.numeric(dimnames(
                              cmp.tree$frame)[[1]]) == i,  ]$n *
                              cmp.tree$frame[as.numeric(dimnames(
                              cmp.tree$frame)[[1]]) == i,  ]$yprob)
                  }
#
# See each node to check where the variable & the split rule is changed.
#
#     1: No change
#     2: Nodes having a number which is greater than the total number
#        of nodes in the original tree
#     3: Variable is changed
#     4: Split rule is changed.  Variable is the same.  Components of
#        child nodes are changed if X is continuous.
#     5: Split rule is changed.  Variable is the same.  Components of
#        child nodes are the same (for continuous X).
#

#
# Put 2 if the node number is greater than the maximum node number of
#    the original tree.  These nodes will not be checked later.
#

                  if(nrow(cmp) > nrow(org)) {
                        for(j in (nrow(org) + 1):nrow(cmp)) {
                          cmp[j, 5] <- 2
                        }
                  }
                  for(i in 1:nrow(org)) {
#
                        if(cmp[i, 5] != "1" && cmp[i, 5] != "2") {
#
# Put 1 if the splitting rule is the same as that of the original tree.
#
                          if(cmp[i, 2] == org[i, 2]) {
                            difference <- 0
                            if(cmp[i, 3] == org[i, 3]) {
                              cmp[i, 5] <- 1
                            }
                            else {
#
# Put "4", if the variable is the same but the splitting value is
#    different.  For continuous X, if components of the child nodes are
#    the same, put "5." ...
#
                              if(is.x.cat == F) {
                                cmp[i, 5] <- 5
                                difference <- sum(abs(as.numeric(org[i *
                                    2, 5:(4 + nom.classes)]) -
                                    as.numeric(cmp[i * 2, 7:(6 +
```

```
                                    nom.classes)])))
                                cmp[i, 6] <- difference/
                                    sum(as.numeric(org[i, 5:(4 +
                                    nom.classes)]))
                                       # fraction of cases falling in to
                                       #    that node and switching
                                       #    between the child nodes
                                }
#
                                if(is.x.cat == T || difference > 0) {
                                  cmp[i, 5] <- 4  #
#
# ... and check child nodes and put "1" (which means "checked,") not to
#    check them later.
#
                                  depth1 <- floor(log(i, base = 2)) + 1
                                  if(depth > depth1) {
                                    for(k in 1:(depth - depth1)) {
                                      for(j in 1:2^k) {
                                        cmp[(i * 2^k + j - 1), 5] <- 1
                                      }
                                    }
                                  }
                                }
                              }
                              else {
#
# Put "3" if the variable is different...
#
                                cmp[i, 5] <- 3      #
#
# ... and check child nodes and put "1" (which means "checked,") not to
#    check them later.
#
                                depth1 <- floor(log(i, base = 2)) + 1
                                if(depth > depth1) {
                                  for(k in 1:(depth - depth1)) {
                                    for(j in 1:2^k) {
                                      cmp[(i * 2^k + j - 1), 5] <- 1
                                    }
                                  }
                                }
                              }
                            }
                          }
#
# Calculate the total change in tree topology.
#
                          mot <- -1
                          cnode.var <- NULL # change in variable
                          cnode.spl <- NULL # change in splitting rule
                          cd.var <- NULL    # depth where variable is changed
                          cd.spl <- NULL    # depth where split. val. is changed
#                         cnode.nom <- NULL #
                          mot.var <- 0      # change in topology caused by
                                            #    change in variable
```

```
                    mot.spl <- 0        # change in topology caused by
                                        #    change in split. rule

                    if(nrow(data.frame(cmp)[cmp[, 5] == "1",   , drop = F])
                       == nrow(cmp)) {
#
# If there is no change, total change in topology = 0.
#
                          mot <- 0
                    }
                    else {

#
# Find the node numbers where the variable is changed, if any.
#

                          if(nrow(data.frame(cmp)[as.numeric(cmp[, 5]) ==
                               3,   , drop = F]) > 0)
                            cnode.var <- as.numeric(cmp[as.numeric(cmp[,
                               5]) == 3,   , drop = F][, 1])

#
# Find the node numbers where the splitting value is changed, if any.
#    In the case of continuous X, get the fraction of cases switched
#    between child nodes (cnode switched).
#

                          if(nrow(data.frame(cmp)[as.numeric(cmp[, 5]) ==
                               4,   , drop = F]) > 0) {
                            cnode.spl <- as.numeric(cmp[as.numeric(cmp[,
                               5]) == 4,   , drop = F][, 1])
                            if(is.x.cat == F) {
                              cnode.switched <- as.numeric(cmp[as.numeric(
                                cmp[, 5]) == 4,   , drop = F][, 6])
                            }
                            else {
                              cnode.switched <- 1
                            }
                          }
                    }
                    if(mot != 0) {

#
# Calculate the change in topology due to the change in the splitting
#    variables, ...
#
                          if(length(cnode.var) > 0) {
                            cd.var <- floor(log(cnode.var, 2)) + 1
                            mot.var <- sum(2^(depth - cd.var + 1) -
                               1)/(2^depth - 1)
                          }
#
# ... and calculate the change in topology due to the change in the
#    splitting values...
#
                          if(length(cnode.spl) > 0) {
                            cd.spl <- floor(log(cnode.spl, 2)) + 1
```

```
                                mot.spl <- sum(2^(depth - cd.spl + 1) - 1 - (1
                                    - cnode.switched))/(2^depth - 1)
                            }
                        }
#
# Then, compute the total change in tree topology.
#
                        mot <- mot.var + mot.spl
                        mot.vec[case] <- mot
                }
#
# Store the result for each perturbation rate.
#
                measure.topology[perturb] <- mean(mot.vec)
                misclass.rate[perturb] <- mean(misclass.vec)
                mean.depth[perturb] <- mean(depth.vec)
                mean.nodes[perturb] <- mean(nodes.vec)
                top.std[perturb] <- sqrt(var(mot.vec))
                misclass.std[perturb] <- sqrt(var(misclass.vec))
                depth.std[perturb] <- sqrt(var(depth.vec))
                nodes.std[perturb] <- sqrt(var(nodes.vec))
        }
#
# Output the results.
#
    result <- data.frame(cbind(per, measure.topology, top.std,
            misclass.rate, misclass.std, mean.depth, depth.std,
            mean.nodes, nodes.std))
        names(result) <- c("perturb", "topology", "top.std", "misclass",
            "m.std", "depth", "d.std", "nodes", "n.std")
        used.seed <- cbind(per, used.seed)
        result.all <- list(call = match.call(), first.seed = used.seed,
            result = result)
        result.all
}
```

C. FUNCTION FOR PERTURBING CATEGORICAL VALUE

```
function(y, prob = p)
{
#
# Samuel E. Buttery
#
# Chy : change y according to our scheme
#
# Arguments: y     : data to be changed
#            prob : prob. of any one item being changed
#
# Step 1: compute priors with table(); save the names of the priors.
# These names are the set of values of y.
#
      priors <- table(y)/length(y)
      values <- names(priors) #
#
# Figure out which y's need to be changed.
#
      changers <- sample(c(T, F), size = length(y), replace = T,
            prob = c(prob, 1 - prob))      #
#
# For each unique value of y, find the ones that need to be changed
# just within that subset.
#
      old.y <- y
      for(i in 1:length(priors)) {
            new.probs <- priors[names(priors) != values[i]]
            new.probs <- new.probs/sum(new.probs)
            relevant <- changers & old.y == values[i]
            y[relevant] <- sample(names(new.probs), size =
                  sum(relevant), replace = T, prob = new.probs)
      }
      return(y)
}
```

LIST OF REFERENCES

Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. (1984), *Classification and Regression Trees*, Monterey, CA: Wadsworth and Brooks.

Chatterjee, S., Hadi, A. S. (1988), *Sensitivity Analysis in Linear Regression*, New York: John Wiley & Sons.

Michie, D., Spiegelhalter, D. J. and Taylor, C. C. (1994), *Machine Learning, Neural and Statistical Classification*, New York: Ellis Horwood.

Miclet, L. (1986), *Structural Methods in Pattern Recognition*, New York: Springer-Verlag.

Blake, C.L. & Merz, C.J. (1998). UCI Repository of machine learning databases [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.

Venables, W. N. and Ripley, B. D. (1994), *Modern Applied Statistics with S-Plus*, New York: Springer-Verlag.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey California 93943-5101

3. Professor Samuel E. Buttrey 1
   Code OR/SB
   Naval Postgraduate School
   Monterey California 93943-5002

4. Professor Robert A. Koyak 1
   Code OR/KR
   Naval Postgraduate School
   Monterey California 93943-5002

5. Operations Evaluation Office, MSO 1
   Japan Defense Agency
   9-7-45 Akasaka, Minato-ku
   Tokyo 107-8513, Japan

6. Izumi Kobayashi 1
   Code 30
   Naval Postgraduate School
   Monterey California 93943-5002